HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Electronics, Communications and Automation
Department of Automation and Systems Technology

LULEÅ UNIVERSITY OF TECHNOLOGY

Qu Zengcai

Intelligent Diving Control of a Lagrangian Type of Underwater Robot

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology

Espoo, 07.29.2009

Supervisors:

Docent Mika Vainio D.Sc.(Tech.), Helsinki University of Technology

Professor Kalevi Hyyppä D.Sc., Luleå University of Technology

Instructor:

Eemeli Aro, M.Sc.(Tech.)
Helsinki University of Technology

Acknowledgements

First of all, I would like to acknowledge the committee of Erasmus Mundus SpaceMaster program for giving me the opportunity to have the two years of Master's study around Europe, as well as the financial support of the Erasmus Mundus scholarship. I want to give my kind acknowledgement to Mr. **Sven Molin**, who helped me greatly and made a great contribution to the SpaceMaster program.

I would like to gratefully acknowledge Professor **Aarne Halme**, department of Automation and Systems Technology, TKK, for giving me the opportunity to study robotics in Helsinki University of Technology, and for all the kind guidance and help during my study and research. I appreciate Prof **Kalevi Hyyppä**, Lulea University of Technology, for the review and comments to my thesis work and report.

I want to thank Mr. **Eemeli Aro** who pointed me in the right direction whenever I faced problems with my project. The SWARM simulator written by Mr. Aro is an important tool for my thesis project, which was of much assistance in my control design. I appreciate him for all his kind and rapid answers to my questions and for the guidance he gave me in the Baltic environments study and control design.

I would like to acknowledge Mr. **Mika Vainio** for all the kind and patient help during my whole thesis project, and for the arrangement of the project work, the communication with the marine institute and the preparations for the practical diving test. Additionally, I wish to thank Mr. Vainio for the guidance and literature recommendation.

I would like to gratefully acknowledge Mr. **Tomi Ylikorpi** and Ms. **Anja Hänninen**, Department of Automation and System Technology, TKK. They helped me a lot during my study in Finland, and handled the arrangements for the SpaceMaster

program at TKK. Mr. Ylikorpi gave me a lot of encouragement in the beginning of my thesis project.

Mr. **William Martin**, Department of Networking and Communications, TKK, gave me a lot help with my language and thesis writing, and supported me greatly with my oral defense.

I would like to acknowledge all my **SpaceMaster** colleagues, who gave me a lot of beautiful memories during the two years of study, especially thanks the classmates in Finland for the discussion and advising in my courses and thesis.

Finally, I want to give my best appreciation to my parent, I wish them health and happiness forever!

Espoo, July 29, 2009

Qu Zengcai

Helsinki University of Technology Abstract of the Master's Thesis

Author:	Qu Zengcai
Title of the thesis:	Intelligent Diving Control of a Lagrangian Type of Underwater Robot
Date:	July 29, 2009 **Number of pages:** 93
Faculty:	Faculty of Electronics, Communications and Automation
Department:	Automation and Systems Technology
Program:	Master's Degree Programme in Space Science and Technology
Professorship:	Automation Technology (Aut-84)
Supervisors:	Docent Mika Vainio D.Sc.(Tech.) (TKK)
	Professor Kalevi Hyyppä (LTU)
Instructor:	Eemeli Aro, M.Sc.(Tech.) (TKK)

The SWARM underwater system, which consists of multiple, homogenous robots, is a project for monitoring the Baltic Sea environments. The unit of the system is a Lagrangian type robot which can control its depth by changing the buoyancy using a piston engine, and moves otherwise freely with the sea water flows. The depth control of the robots is very important for the performance of the whole system.

This thesis provides an energy optimal diving control algorithm for the SWARM robots to improve the performance and energy consumption of the diving process. Due to the nonlinear environment and dynamics, most of the classical theories may not achieve good results and consume much energy. The environment based depth control algorithm estimates the water density by a density model, which is built from previous information or measurement. By reducing the piston movements, much energy can be saved than the previous control algorithm.

The Baltic environments are studied and the physical water properties including salinity, temperature, pressure and density are analyzed. Result proves the availability of the seawater properties model in diving control. Different methods of measuring seawater density and updating the environment model are evaluated, and the environment based control algorithm is tested in the simulator. Error and energy consumption are analyzed and the result shows the diving performance is improved significantly.

Keywords: underwater robot, swarm system, diving control, optimal energy consumption, environment model, environment based control

Contents

List of Tables

List of Figures

Symbols and Abbreviations

TKK	Teknillinen korkeakoulu, Helsinki University of Technology
AUV	Autonomous Underwater Vehicle
ROV	Remotely Operated Vehicles
VBS	Variable Buoyancy System
SWARM	a multi-probe underwater system
CTD	A sensor measures Conductivity, Temperature and Depth
C	Conductivity
S	Salinity
T	Temperature
ρ_w	Density of seawater
ρ_r	Density of SWARM robot
z	Depth
V	Volume
v	Velocity
a	Acceleration
DOF	Degree of freedom
SMC	Sliding Mode Controller

Chapter 1

Introduction

"Underwater men shall walk, shall ride, shall sleep and shall talk"

--Martha Shipton (1488-1561)

Nowadays, people's imaginations are captured by the exploration of space, but they might forget that there are still many places on this planet that have not yet been fully explored, especially the ocean. As the saying goes..."water, water, everywhere." 70.8% of the Earth surface is covered by ocean, and there are also a great many of lakes, rivers, and canyons filled with water around us. More and more research and work needs to be operated in underwater environments which are not suited to humans. Therefore, underwater robots are important and potential tools for underwater exploration and research.

Underwater robots are autonomous agents operating in underwater environments. There are two main categories of underwater robot: autonomous underwater vehicle (AUV) and remotely operated vehicle (ROV). The difference of AUV and ROV is that AUV is controlled automatically by on-board computers and can work independently without connection to the surface. ROV, on the other hand, is controlled or remote controlled by a human operator from a cable or wireless communication on a ship or on the ground. In this thesis, SWARM float is a specific AUV, which is also known as Variable Buoyancy System (VBS). This kind of robot controls its depth by variable buoyancy and in the meanwhile drifts with the water current freely.

The dynamics and control of underwater robots are quite different from the ways utilized by other robots work in more general environments due to the complicated forces of the water. An underwater robot may suffer from many different forces including buoyancy, drag, gravity and flow forces, which all depend on water density, sea current, liquid viscosity, robot surface, shape, velocity, and etc. An underwater robot may float at a certain depth or density, dive from one depth to another, swim in the water, walk or roll on the bottom of the seabed. There are many unknown forces that can change the position of the robot undesirably, thus, an underwater robot has to sense and control its position and orientation actively. The control performance relies heavily upon the quality of the sensing, and the positioning, thus controlling an underwater robot is a major challenge due to the complicated environments and the limitations of the sensors.

This thesis project works on the SWARM robots which are Lagrangian type underwater drifters floating in the Baltic Sea. Lagrangian drifter refers to the Lagrangian motion of the robot with the seawater current. The buoyancy of the robot is controlled by a piston engine moved by a motor so that the robot can dive to or stay at a certain depth in the water, the float uses a CTD sensor to measure the Conductivity (salinity), Temperature and Depth (pressure) of Baltic Sea. The sensors and actuators are very limited so that an optimal diving control algorithm is required for the SWARM system. The thesis solves the diving control problems and tries to find an optimal control algorithm considering the energy consumption, error, stability, and system performance.

The paper is organized into 7 chapters: Chapter 1 here offers a brief introduction of the thesis project; Chapter 2 reviews the state-of-the-art of underwater robotics, and introduces some examples of underwater projects similar to SWARM project, evaluates the diving control methods; Chapter 3 goes into details of the SWARM project, introduces the hardware and software of the SWARM float, as well as the SWARM simulator; Chapter 4 is the result of this thesis study of the underwater environments, specifically the Baltic Seawater physical properties; Chapter 5 presents the dynamics of the underwater robot and then proposes the control method, evaluates the new control theory with the old controller from the result of the simulator test; Chapter 6 proposes the plan of the practical tests in the Baltic Sea which will not be

included in this report. Chapter 7 is the conclusion of the thesis project and the result of the control design.

Declaration:

My contributions to this thesis are studying and modeling the Baltic environments, raw data processing, the design and testing of the environment based depth control algorithm, the diving test code, diving data processing, water density calculation and measurement methods evaluation, dynamic parameters test, the effect of the drag coefficient, and the problems and solutions in the depth control algorithm, etc.

The SWARM robot hardware introduced in chapter 3 is designed by **The SWARM consortium**, and the simulation platform introduced in chapter 3 is written by **Eemeli Aro**, Automation and System department, TKK.

Chapter 2

Literature Review

2.1 State-of-the-art

The history of underwater robot technology can date back to the first torpedo designed in 1866. In the first torpedo prototype, simple hydrostatic valve acting directly on the elevator to hold the torpedo at a pre-determined depth. In practice however the depth control was erratic, sometimes running along the surface, sometimes plummeting to the depths and often acting like a porpoise, but this can be considered to be the pioneer of underwater robot control.[1] The first “true” underwater vehicle was developed in the late 1950s by Stan Murphy, Bob Francois, and Terry Ewart of the Applied Physics Laboratory at the University of Washington. The purpose of this vehicle is to obtain oceanographic data along precise trajectories and under ice.[2] The work in University of Washington led to the development and operation of the Self Propelled Underwater Research Vehicle (SPURV) in the 1960s.[3] In 1983 the Advanced Unmanned Search System (AUSS) was developed by SPAWAR in response to the sinking of the USS Thresher, the USS Scorpion, and the H bomb loss of Palomares.[4] In 1990s the underwater robot technology developed rapidly, and operated world-wide, there were some leading underwater projects such as Odyssey (MIT), Autonomous Benthic Explorer (Woods Hole Oceanographic Institution--WHOI), Theseus (International Submarines Engineering, Ltd.) and Remote Environmental Measuring UnitS--REMUS (WHOI).[5] After 2000, more and more underwater project are in research or commercial use, which covers a large area of exploration, wrecking, military, and fishery.

Underwater robots are used for many purposes, including monitoring water pollution, biology research, shipwrecks investigation, as well as the exploration of other ocean environments. Underwater robots are essential tools of modern ocean exploration. As a branch of robot technology, underwater robotics has developed rapidly over recent years.

Currently, the department of Automation and System Technology at TKK takes part in three underwater robot projects: SUBMAR, DAMOCLES and SWARM: SUBMAR is an intelligent, autonomous miniature robots system for the monitoring of liquid processes, the units are ball kind of robots which controls their vertical movements in the liquid and can do 3D measurement thanks to the active navigation and positioning systems; DAMOCLES is a project for developing arctic modeling and observing capabilities for long-term environmental studies; and SWARM is a Lagrangian autonomous underwater multi-probe system for coastal area/shallow water monitoring.

Water is the most important resource on our planet, and the ocean environment is not fully explored yet, the research of ocean will be an important subject for human in the 21st century. The development of the robotics technology will give a bright future to the ocean exploration, the underwater robots will be more and more widely used in this area. Underwater robot technology is rapidly evolving in a highly competitive market.

The commercial underwater robots will help people in ocean research, rescue operations, biological monitoring for marine pollution, searching wrecked ship, underwater resources exploitation, deep-sea environments, geognosy and biology exploration, as well as military. The similar technology and application can even be used in other liquid such like oil and mud, and also in space explore mission for there is some evidence shows that there might exist liquid on the other planet. Multi function vehicles will work under the ocean water for different missions, help human to exploit the ocean.

2.2 Typical underwater projects and diving method

There are many different kinds of underwater robots, which are designed for different purposes. The diving control methods are different depend on the operation environment and performance. This part will introduce several underwater robots, most of which are variable buoyancy systems (VBS) and similar with SWARM project, the different diving system designs and control methods will be compared and evaluated. Although DEPTHX is not a variable buoyancy system and quite different from SWARM project, the diving system and control are very unique and different, it is introduced to compare with variable buoyancy systems.

2.2.1 ARGO float – International integrated ocean observing system

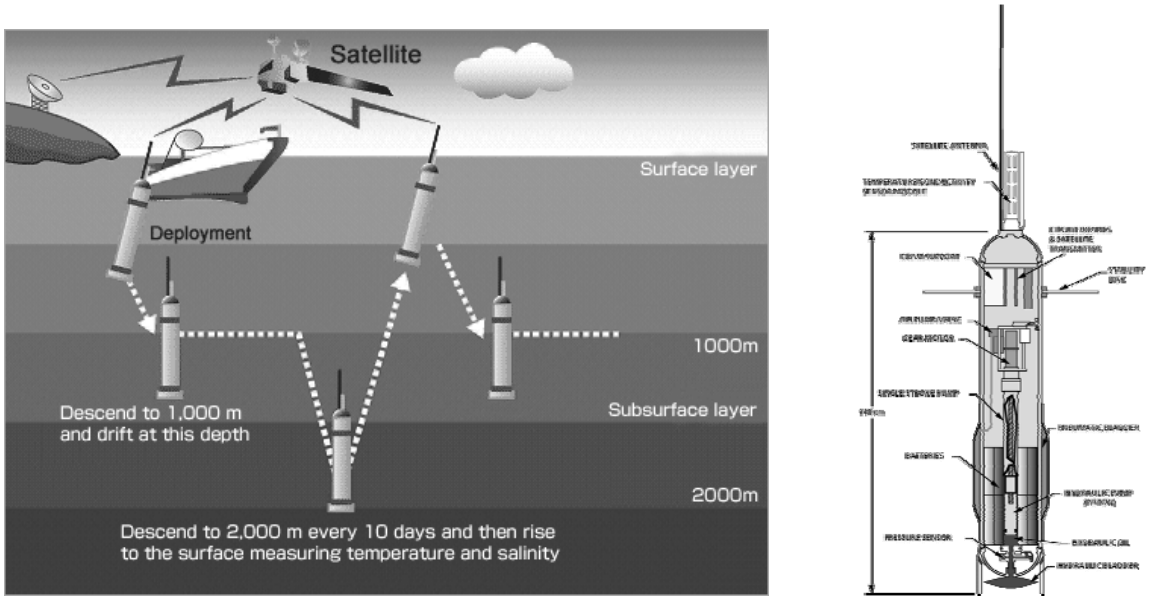

Fig. 1. Argo mission demonstration and the cutaway diagram of the float

The Argo project is an international collaboration operated by 50 agencies from 26 countries. Argo floats can dive to 2000 meters depth and collect high-quality measurements of the seawater including temperature and salinity.[6] The floats drift at certain depths by being neutrally buoyant controlled by a hydraulic pump or piston. Figure 1 presents an example mission demonstration and cutaway mechanism diagram of the Argo float.

From the measurements of the floats, information about sea currents, temperature, salinity and mass redistribution can be inferred, which are useful for ocean research

and atmosphere forecast. The floats dive periodically and once they pop up to the surface, the measurements data can be sent to the Iridium satellite through the antenna at the top of the float.

Figure 2 shows the array of Argo floats in June 2009 distributed over the global oceans, consists of more than three thousand floats. The final array of 3000 floats can provide 100,000 T/S (temperature /salinity) profiles and velocity measurements per year at an average 3-degree latitude and longitude spacing. The Argo float can reach a depth of 2000m to complete a mission cycle every 10 days, the individual float has a lifetime of about 4-5 years.[7]

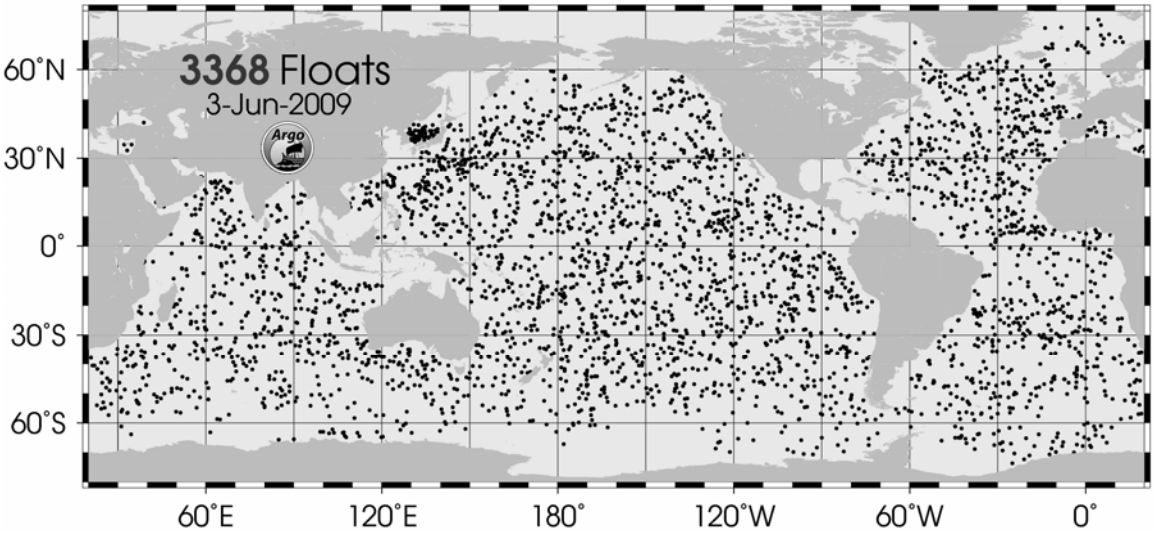

Fig. 2. Array of Argo floats on 03.June.2009

2.2.2 RAFOS float – sound fixing and ranging

RAFOS is the backwards spelling of SOFAR (SOund Fixing And Ranging), which refers to the way these floats track the motion of water in the ocean. The floats are used for mapping the mean currents over a wide area and statistical studies of dispersion and mixing, especially for the study of eddies and boundary currents. The project uses acoustic tracking to provide relative trajectories of the floats with position fixes several times each day.[8] The tracking information can create a map of the float trajectories so that the current flow can be deduced and analyzed.

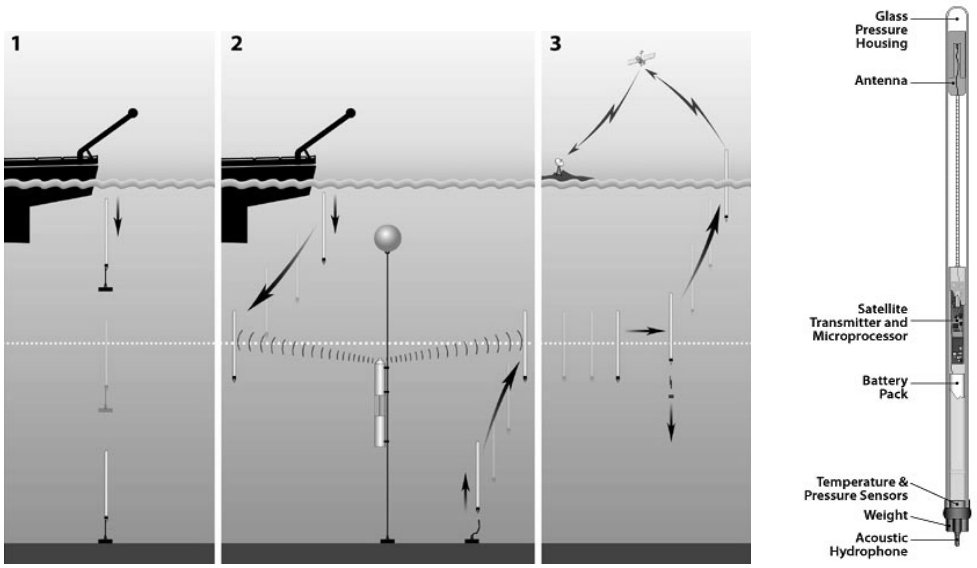

Fig. 3. RAFOS diving demonstration and the cutaway diagram of the float

As figure 3 shows, In the beginning of a mission, a drifting RAFOS float is dropped into seawater from a boat and sinks by the additional weight of ballast (1), during the diving process, it drifts with current flow, listens and records sound signals from stationary acoustic beacons (2). When the mission is ended, it drops the ballast weights, rises to the surface, and communicates with the orbiting satellites (3). The mission can be up to two years in duration. In figure 3, there is a cutaway diagram of the float which shows the hardware and mechanical design. The floats can carry multiple sensors for measuring seawater properties, including pressure, temperature, and dissolved oxygen.

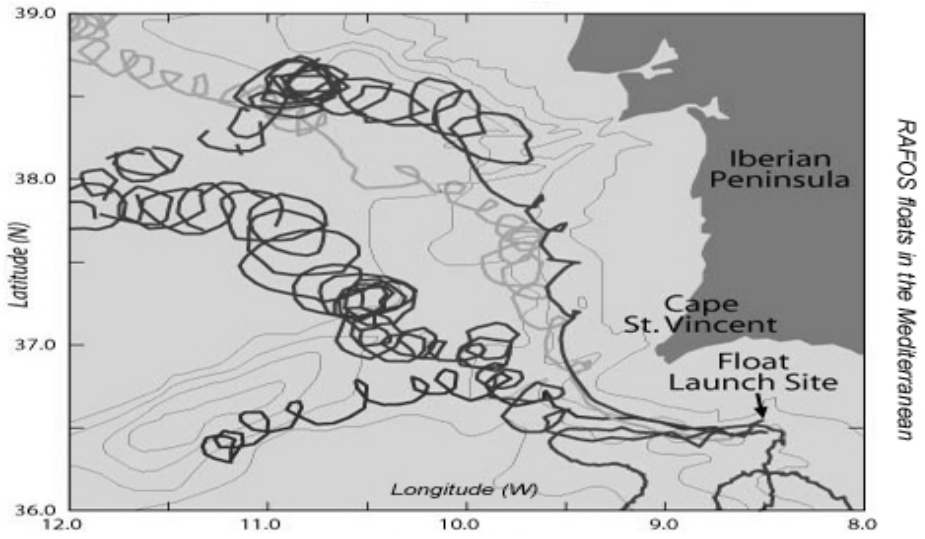

Fig. 4. RAFOS floats in the Atlantic Ocean

The above figure 4 shows a mission of RAFOS float, four floats were released in the Atlantic, and the trajectories of four RAFOS floats were tracked as the curves shows. They followed the Mediterranean eddy, which is also known as "meddies" off the southwestern corner of the Iberian Peninsula.[8]

2.2.3 SEAGLIDER –Autonomous underwater slider vehicle

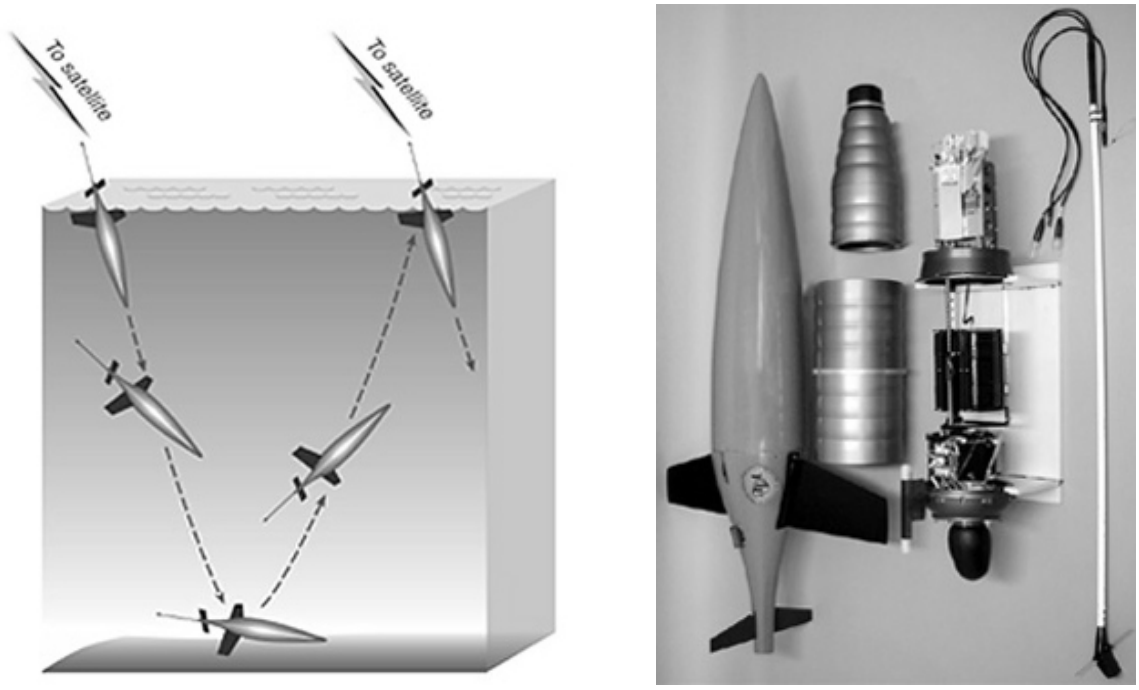

Fig. 5. Autonomous Underwater Slider Vehicle

SEAGLIDER is an underwater exploring project of the University of Washington. The Seaglider can measure conductivity and temperature by a Seabird Electronics (SBE) conductivity and temperature sensor pair, pressure by a pressure sensor and sea bottom depth by an altimeter transducer. In addition, it has an oxygen sensor, and an optical backscatter and chlorophyll fluoremeter. Other sensors can be added to Seaglider by the user depend on different purposes.

SEAGLIDER glides slowly through the water by the wing lift and the robot is propelled by buoyancy control alternately dive and climb along slanting glide paths as figure 5 shows. The slider dead reckons under water between GPS navigation fixes obtained at the sea surface to glide through a sequence of programmed targets. In every diving cycle, the glider collect measurements and once it climbs to the sea

surface, it transmits measurement data to the satellite using the antenna via satellite data telemetry. [9]

2.2.4 DEPTHX – The deep Phreatic thermal explorer

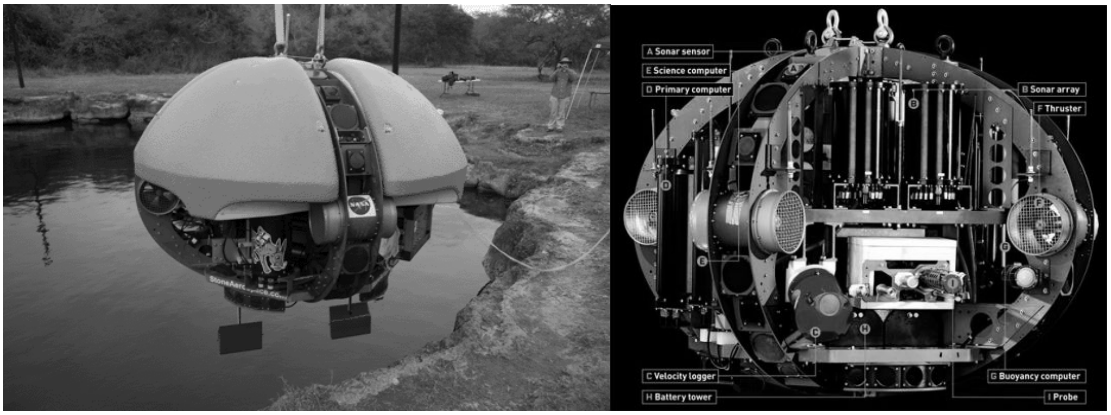

Fig. 6. DEPTHX underwater robot and internal structure.

DEPTHX is an underwater project by NASA designed for exploring and mapping the physical shape of underwater spaces. The robot creates three-dimensional maps of unexplored caves, flooded caverns and mines using sonars sensors, models the environmental parameters, and collects specimen. One of the exploration missions is mapping the deepest flooded sink hole in the world, Zacatón cenote in central Mexico and investigating the existent unique organisms in the cenote. The official definition of the project goal is described as: “Exploration of an unmapped underwater caves and tunnels; three-dimensional mapping of volumetric extent; modeling of environmental parameters and their gradients; characterization of localized site to identify region candidates for biological investigation; automated image and data collection with in-situ analysis for adaptive sampling.”[10]

As shown in figure 6, DEPTHX is an egg-shaped underwater robot which has a long axis of 4.26 meters, a short axis of 3.04 m, and a weight of 1.3 metric tons. The main hardware of DEPTHX consists of 36 onboard computers, 56 sonar sensors and 6 thrusters.[10] The sonars and thrusters are mounted in different directions around the robot fixed on the steel frame of the vehicle. It can dive and hover in a cave or cenote up to 1000 meters deep by the thrusters without any external commands. The robot uses inertial measurements and sonars to map of cave walls and do localization and navigation in the cave. The SLAM algorithm software for DEPTHX uses the sonar

sensors to create a three-dimensional shape of the around environments as a whole instead of looking at discrete objects.[11] As the robot diving in the cave, it also spins around about once per minute, and the sonars fire up to the surrounding four times per second, allowing the beams to “paint in” the cave walls.[11]

2.2.5 Diving methods and evaluation

The four typical robots are selected because they are using different kind of diving methods. Table 1 presents and compares the different diving methods of those underwater robots.

Table 1. Diving control methods

Method	Implement	Project	Introduction
Variable volume (Buoyancy)	Piston or Pump	Argo, SWARM	Vertical depth is controlled by changing the buoyancy and the robot drifts freely in horizontal with current flow. The robot is controlled by a pump or piston engine, which change the volume of the robot so that the buoyancy changes
Variable mass (Buoyancy)	Suspender	RAFOS	Diving by hanging and throwing suspender. This kind of float cannot change their depth freely but drifts in an initialized density. The float consumes very little energy so that the mission can last for a long duration.
Propulsion	Thruster	DEPTHX	Diving by the propulsion from thruster, The robot or float is designed and calibrated neutral buoyant, the vertical movement is propelled by thrusters. It needs much energy consumption but has high mobility and speed.

Lift force	Wing	Seaglider	The float can glide in the water using the lift force by a pair of wing. The vertical forces can generate horizontal movement. It makes the robot glide in the diving process.
Other diving design			

2.3 The challenges of controlling underwater robots

The dynamics of an underwater robot depends on the surrounding environments greatly. The forces from surrounding seawater affect the slightest nuance of the vehicle motion, small errors may cause control failure or change the vehicle behaviors undesirably. The construction and control design of an underwater robot is fully preoccupied with environmental considerations. For the SWARM robot, there are many challenges of diving control using a classical controller:

1. There are a lot of errors cannot be avoided, such as the total mass of the float, the volume of the piston, the sensor measurement error, and the drag coefficient, etc. All of these measurement errors may cause large depth error as well as fatal failure.
2. The drag force and buoyancy are nonlinear and change with the state of the robot. Drag force depends on the water drag coefficient, cross sectional area, and also depends on the speed of the robot. Buoyancy is nonlinear and changes along depth due to the nonlinear seawater density profile.
3. Sensors and actuator (piston engine) have delay. The CTD measurements interval is about 1.4 seconds and the piston cannot move to the desired position on the instant. Late perception and reaction are always dangerous for an underwater robot in the real world.

Pressure sensors and sonars are mostly used for the underwater robot control. In clear water, robots can use cameras for localization and navigation, but in the deep ocean, a camera is not available because of low illumination and the image is disturbed by noise of the particulates in the seawater. Depth can be measured by static pressure. Transponder systems provide drift-free position measurements once they have been

put in place. Therefore, pressure sensors are mostly used for depth control and transponder systems are always used for localization and navigation.

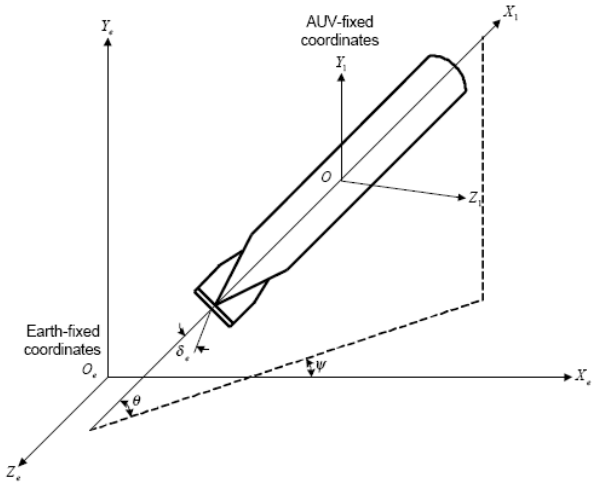

Fig. 7. Reference frames of 6DOF underwater vehicles

As mentioned in the second chapter, the nonlinearity of seawater properties makes the control of an underwater robot very challenging. The control of a six degree of freedom underwater vehicle is the most difficult, which needs not only to control the depth but also the attitude, balance, and orientation. Figure 7 shows the reference frames of a six degree of freedom underwater vehicle. The unpredictable, time variant, nonlinearity disturbances can change the dynamics and states of the vehicle undesirably, fast and active reaction is highly needed. The controller should be capable of self-adjusting control parameters.[12]

Generally speaking, the diving control of variable buoyancy system is much simpler than a 6DOF underwater vehicle, because there is only one degree of freedom—vertical depth.

2.4 Classical control algorithms of underwater robots

A simple PID controller is not good enough to control a 6DOF AUV, the key problem in applying PID is the tuning up of three parameters (proportional coefficient, integral time and differential time).[13] Various advanced control schemes for underwater robots have been proposed in the literature as some of them are summarized below.

Sliding mode control (SMC): SMC is said to be the most widely researched approach for controlling underwater vehicles since the mid 1980s.[14] In control theory, sliding mode control, or SMC, is a form of variable structure control (VSC). It is a nonlinear control method that alters the dynamics of a nonlinear system by application of a high-frequency switching control.[15] This method was first used on underwater vehicle control by Yoerger and Slotine[16]

Adaptive control: Adaptive control involves modifying the control law used by a controller to cope with the fact that the parameters of the system being controlled are slowly time-varying or uncertain.[17] For example, as a variable buoyancy underwater robot, while the robot is diving to deeper depth, the water density will slowly increase, and the robot volume will decrease slightly due to water pressure; we need a control law that adapts itself to such changing conditions. Many researchers use adaptive control combined with neural network.[18]

Nonlinear robust/optimal control: Optimal control theory, an extension of the calculus of variations, is a mathematical optimization method for deriving control policies. One novel nonlinear optimal control example for underwater vehicle control is based on the formulation of nonlinear optimal control problem which leads to the Hamilton-Jacobi-Bellman equation and its approximate numerical solution by iterative application of a Galerkin type method.[19]

Neural network (NN) control: neural networks have attracted many researchers because they can achieve nonlinear control and mapping. The advantage of using neural networks in underwater robot controlling is that the dynamics of the controlled system need not be completely known. This makes NN suitable for underwater vehicle control. Figure 8 shows an example of AUV control using adaptive neural network.[18]

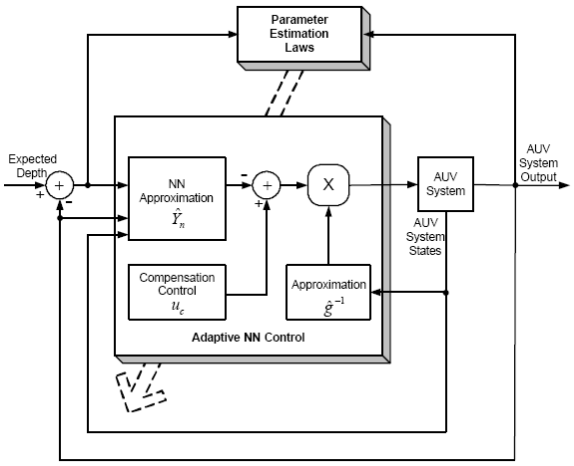

Fig. 8. An example of AUV control using Adaptive Neural Network

Fuzzy logic control: A fuzzy control system is a control system based on fuzzy logic, which is a mathematical system that analyzes analog input values in terms of logical variables between 0 and 1(true and false).[20] For control engineering applications, researchers use fuzzy logic to form a smooth approximation of a nonlinear mapping from system input space to system output space. This makes it suitable for nonlinear system control.[14] Fuzzy logic control for the VBS is based on the experience and knowledge of the human operator. One example is Min Xu and S.M.Smith's adaptive fuzzy logic depth controller for VBS. The controller brings the robot to a desired depth and then restores neutral buoyancy. The control strategy consists of Long Range Depth Control (LRDC) and Short Range Depth Control (SRDC). The LRDC adds robot density to reach a desired terminal velocity and then keep this velocity until the robot reaches the critical depth1. Then the controller reduces the robot density to start decelerating until depth2. Finally, the robot reaches the desired depth with neutral buoyancy by estimation. If the final depth is not close enough to the desired depth, SRDC is used to make one or more adjustments. Fuzzy logic is used to compute the two critical depths.[21]

Chapter 3

SWARM system and Simulator

3.1 SWARM—Underwater multi-probe system

SWARM is a research project supported by the European Commission under the Fifth Framework Program and contributing to the implementation of the key action "Sustainable Marine Ecosystems" within the energy, environment and sustainable Development.[22]

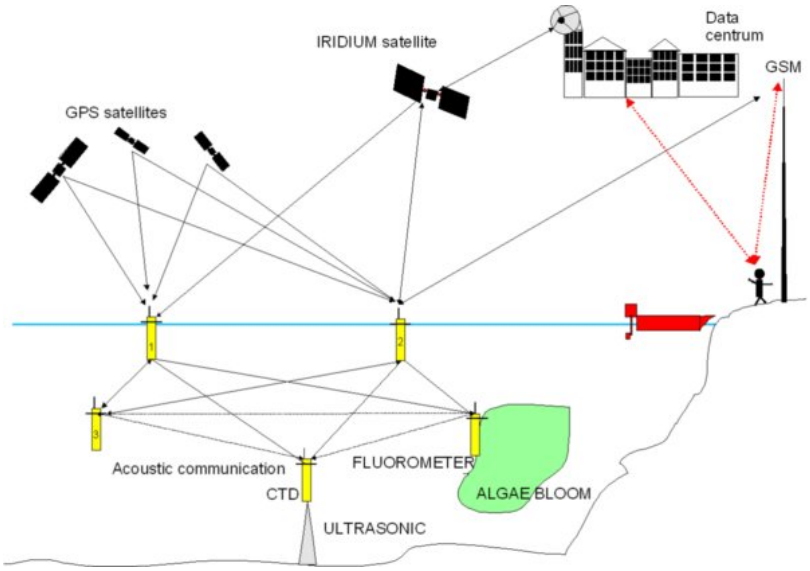

Fig. 9. Demonstration of SWARM underwater robots system

SWARM system is shown in figure 9. The main object of SWARM system is to develop a multi probe detection platform that can measure biological and physical,

chemical, and biological coupling of coastal ecosystem. SWARM system is used especially for the exploration and research of the Baltic Sea.

This project is a multi-robots system and the drifters floating with the sea flow in different depths forming a homogeneous team. A multi agents system can cover a more large area, and the units can share information and cooperate with each other. The communication between the drifters is by ultrasonic. The control architecture is either centralization or distributed.

In SWARM system, each drifter measures the environments around itself and it can send the measurements to the other drifters by which they can build an integrated model of the environments which is important for group level plan and control.

The multi-robots system can build a control network. The drifters float on the surface can get communication from the satellite and ground station, so that it can deliver messages or commands to the other robots which cannot communicate with the satellite and ground station directly.

The SWARM robots float in different depth layers, the units of the system build a network via communication, and the float itself acts as a Lagrangian object which moves freely with the seawater current.

3.2 Hardware introduction of SWARM float

3.2.1 Sensors of SWARM float

SWARM float uses a CTD sensor, an altimeter, a fluoremeter, an acoustic modem, an Iridium satellite modem and a GPS.

A CTD sensor is an instrument to measure the seawater properties including conductivity (or salinity), temperature and pressure (or depth). The SWARM float uses a CTD model of AQUAlogger 210CTD, which has a 8 channels 16-bit acquisition, 2 channels input, RS232 or RS485 and USB 1.0 communication (no power required) and (2M sets)/(number of channels enabled) of non-volatile flash storage.[23] The CTD sensor is the most important sensor for the SWARM drifter, which can measure the conductivity, temperature and depth (pressure). The swarm

float can get a CTD measurement every 1.4 second and the unit in TKK SWARM lab has low conductivity measurement precision(around 20% error), these should be considered as limitations of the diving control algorithm design. The measurement delay, interval and inaccuracy will be discussed in next chapter.

Fig. 10. CTD sensor of SWARM float

Altimeter is used to avoid emergency collision, it can detect the seabed and if the robot is too close to the sea bottom, the altimeter will give emergency command and the robot can give up the diving mission and float up to surface or change to safe depth.

The acoustic modem is used for underwater communication between the floats and the Iridium modem is used for the communication between the float and the satellite. Acoustic signal transmits along “sound channel” which depends on seawater properties such as temperature and salinity.

3.2.2 Diving system

The diving system of SWARM floats uses a piston engine. Generally, the “open sea” floats manufactured use two models of engines: pump engines and piston engines. The SWARM floats designed by MATTEC, ENSIETA and TKK in 2002 use piston engine because the piston engine with new control command is the better choice for the SWARM system. The final float diving system hardware design is shown in the following figure 11.

The float changes its buoyancy by the piston engine, which is located in a cylinder and made gas-tight by piston rings. The piston is driven by a DC motor via a piston rod for the purpose of injecting or ejecting the fluid in the cylinder. In another word, the piston changes its inner volume via the motor rotation, so that the buoyancy can be controlled.

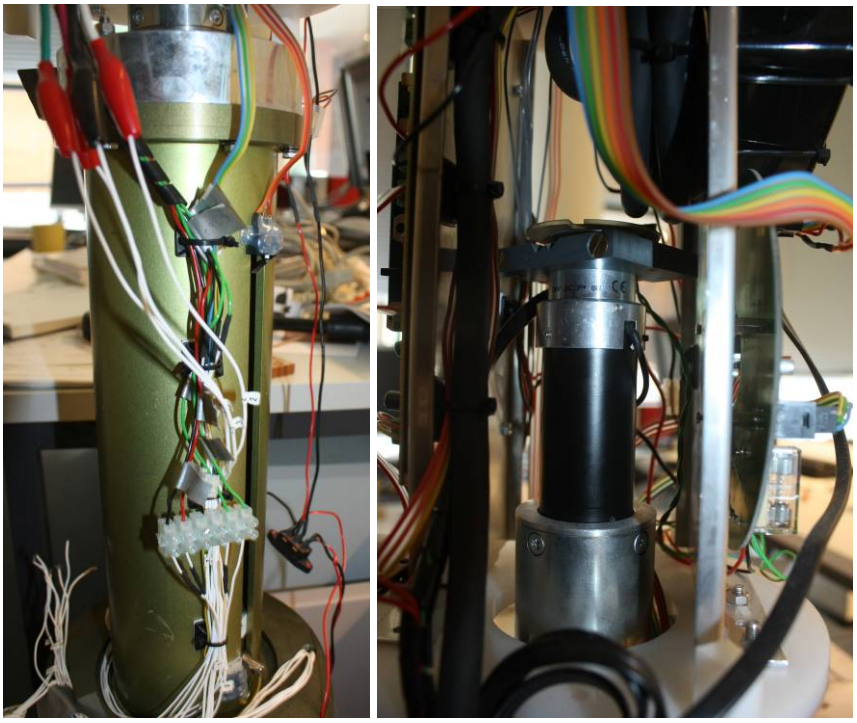

Fig. 11. Piston and motor

There is no capable sensor can measure the inner volume of the piston, however, in the SWARM diving system design, a counter is used to yield the volume. The counter, which is an optical sensor fixed to the motor gear, can read the number of the tours done by the motor. The optical sensor transmits two pulses per tour, and the DC motor is rotating 56.25 tours per second, so the frequency of the counter is 112.5Hz.

The counter is connected to the register T3CON, PIN 5 of the microcontroller. There are two triggers fixed on both sides of the piston cylinder as figure 12 shows, which can stop the motor and reset the counter when the piston reaches the ends of the cylinder, so that the motor will not break itself or the piston. In this kind of counter design, the robot needs to be reset and initialized by moving the piston to the end and resetting the counter to zero before the diving mission, so that the microcontroller can read the motor tours and yield the piston volume.

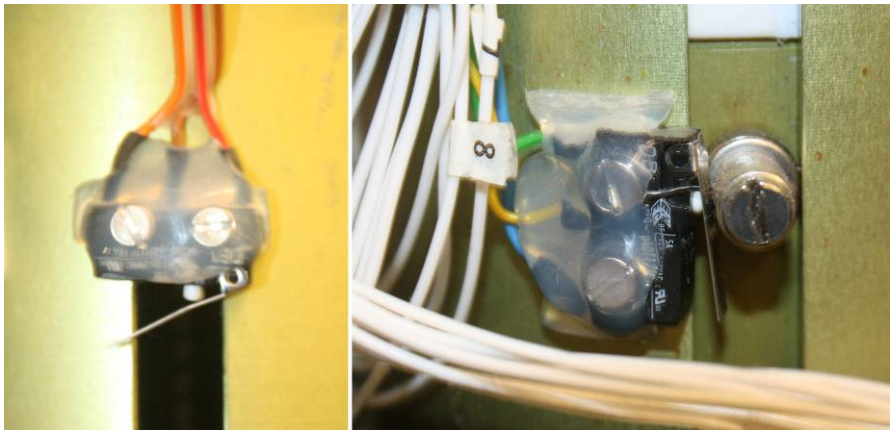

Fig. 12. Triggers

3.2.3 Power system

The DC motor has an input voltage of 24V and an output power of 8w, the power supply comes directly from the power supply board as figure 13 shows.

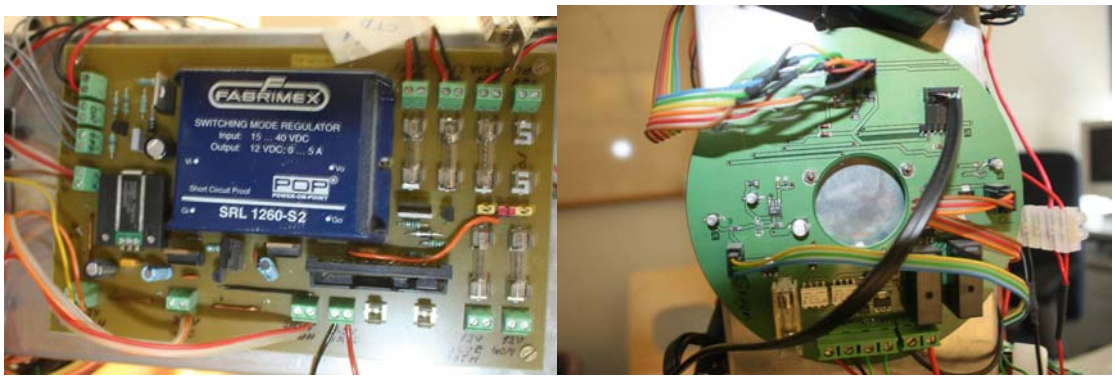

Fig. 13. Power boards

The force to move the piston needs to overcome the friction and water pressure, the water pressure pushes on the piston is large in deep water. The following table shows the energy consumption result of a diving test by ENSIETA in 2004, the diving test

had a trajectory that the float dived to 50m then to 200m and in the end floated up to surface at a speed of 0.1 m/s with an acceleration of 1.10^{-4} m/s^2:[25]

Estimate of energy used during 9700 seconds (in joules):

Piston goes in 1685J	Piston goes out	Total
1475J	2690J	4165J

Piston moving out consumes more energy because the water pressure pushes on the piston, thus the motor needs more torque to push the piston out. As the 2004 diving test report of ENSIETA presented, using their control algorithm, one diving cycle (surface-50m-200m-surface) requires about 5 KJ for a speed of 0.1 m/s with an acceleration of 1.10^{-3} m/s^2, 4 diving cycles per day, therefore a 15 days mission need at least 300KJ energy for the diving system.

3.2.4 Main board

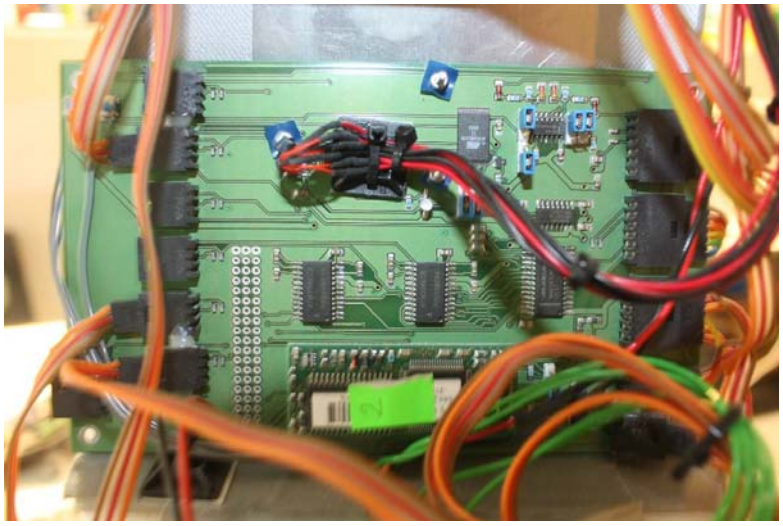

Fig. 14. Main board

The SWARM float has a main board with the microcontroller of Keil C166, on which runs a RTX166 Tiny real-time kernel. Figure 14 shows the main board of the SWARM float.

RTX166 Tiny is a real-time operating system using standard C constructs, which can simultaneously perform multiple functions or tasks. Additional to the C language, the task functions can be declared easily without complex stack and variable frame configuration. The programs need to be compiled with the Keil C166 compiler. The

thesis works mainly on the simulator but not on the robot, and SWARM lab is planning to replace the main board, thus the RTX166 Tiny system will not be introduced in detail here. The following figure shows an overview diagram of RTX166 Tiny kernel.[26]

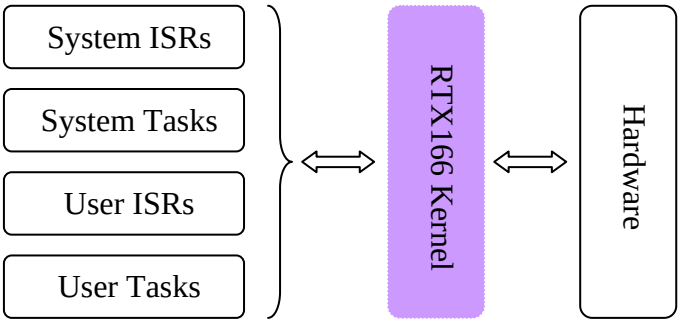

Fig. 15. RTX166 Tiny kernel

3.2.5 Mechanical structure

The final float hardware design is shown in figure 16.

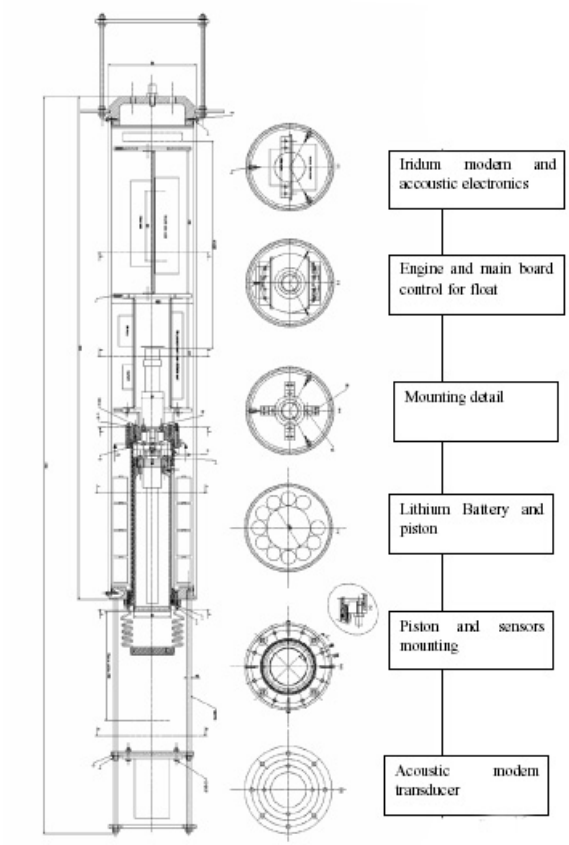

Fig. 16. The mechanical design of SWARM robot

The Iridium modem and antenna are fixed on the top of the float, so that the float can get communication with satellite when it is floating on the sea surface. The computer board and control board are fixed in the upper middle section; piston engine, sensors and battery are fixed in the lower part, this kind of design makes the robot have a heavier mass distribution in the bottom, which enables the float to 'stand' vertically in the water.

The float body is made of metal, and has a height about 2 meters. There are rubber gaskets between the sealing to make the body shell water tight. The SWARM float has a total mass of about 42kg with battery. The float body is shown in figure 17.

Fig. 17. Sketch of the SWARM float body

3.3 A simulation platform for SWARM project

The SWARM simulator, which is written by Eemeli Aero, is a useful tool for the SWARM system design and research. Furthermore, it is a virtual environment for testing the diving code and algorithm. Both numerical and visual results can be obtained from the simulator.

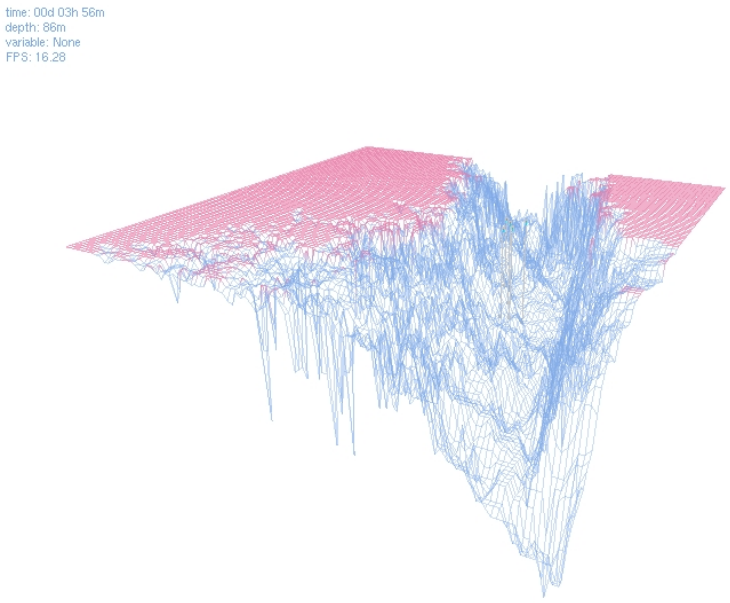

Fig. 18. A screenshot of SWARM simulator

In figure 18 is a captured screenshot of the simulator, where the simulator builds an environment of the Baltic Sea in the area of longitude from 21.8°E to 25°E, Latitude from 59.1°N to 60.7°N. The simulation environment has depth, salinity, temperature, pressure, SSH anomaly, and 3-D current vector information with time.

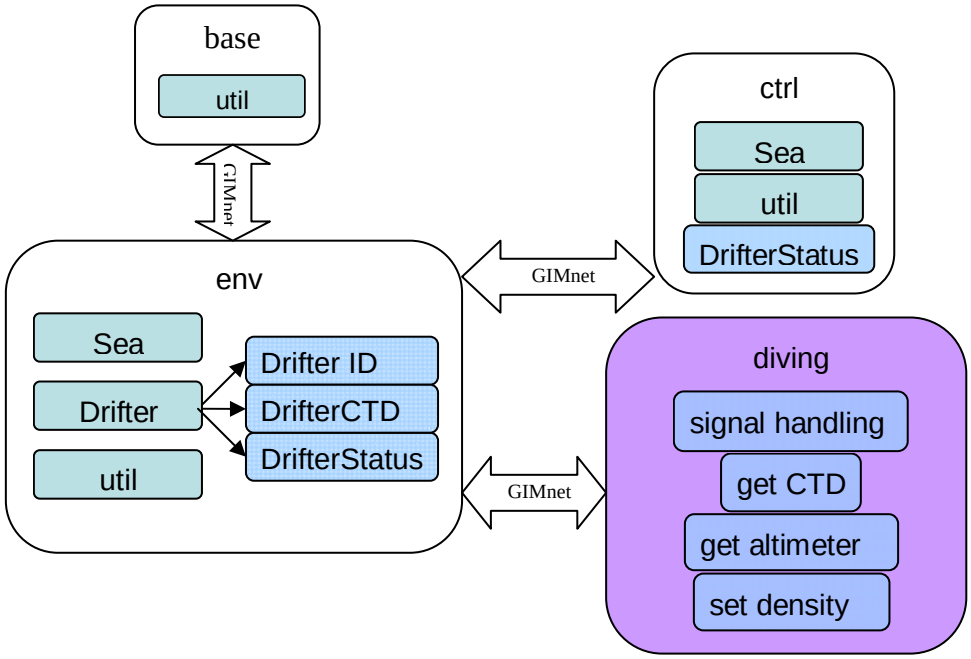

Fig. 19. Architecture of SWARM simulator

Figure 19 is the architecture of simulator. There are four modules: environment, control, base, and diving. The “base” module is not included in the diving simulation,

and the diving control algorithm is implemented in the “diving” module, which has four basic functions: signal handling, get CTD data, read altimeter measurement and set the robot density. The “env” module contains the classes of “Sea” and “Drifter”, and “util”, “util” has a lot of computations such like conversion between depth and pressure, salinity and conductivity and density with CTD. The communication uses GIMnet which is an application-layer communication API. The source code of SWARM simulator needs some libraries including SVL, OpenGL, NetCDF and GIMnet. The instruction of installing the libraries and running the simulator is attached in appendix A.

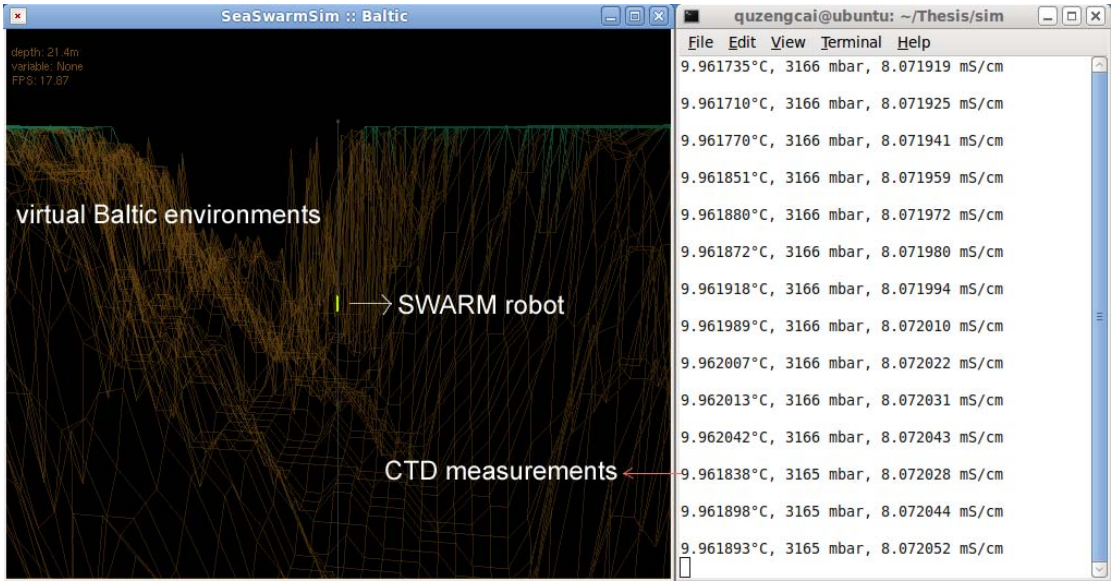

Fig. 20. SWARM robot in the simulator

Both the simulator and the diving control code are in Linux system and using C++, figure 20 shows a screenshot of the simulator, the robot and the diving track is shown in the 3-dimension virtual Baltic environment, and the color of the robot changes with depth. The dynamics and position information are visually represented. The right window is running the diving program, which shows the CTD measurements of the float in the simulator environment.

Chapter 4

Baltic Sea environment research

The study of Baltic Sea environments is an important part of the whole diving control design. The purpose of this chapter is to solve two main questions: how are the underwater physical properties and how can they be used for the diving controlling. The raw data of the Baltic environments are from August 2008, which consist of temperature, salinity, and three dimensional current flows at 21 depths, 17 longitudes, 17 latitudes and 124 times. The raw data are in the netCDF (network Common Data Form) files. To operate and analyze the raw data in Matlab, a netCDF toolbox named “netCDF for Matlab” was used to extract the netCDF file into Matlab matrix files. The seawater properties raw data are from the area of longitude from 21.8°E to 25°E, Latitude from 59.1°N to 60.7°N, where is the Baltic Sea near Helsinki and Tallinn shown in figure 21:

Fig. 21. Map of the Baltic Sea and the area of measurements raw data

The buoyancy of the robot depends on seawater density, thus density is the concerned property need to be analyzed and modeled, but there is no density data, however,

according to the literatures, the water density can be calculated from salinity, temperature and pressure. So the raw data needed for diving control are salinity and temperature. The information of the salinity and temperature netCDF data is shown in table 2, the data are from every 0.2 latitude and longitude, depth from 1.5m (near sea surface) to about 156m, there are totally 752556 data of salinity and temperature.

Table 2. Information of the netCDF raw data.

17 longitudes(°E)	21.8, 22.0, 22.2, 22.4, 22.6, 22.8, 23.0, 23.2, 23.4, 23.6, 23.8, 24.0, 24.2, 24.4, 24.6, 24.8, 25.0
17 latitudes (°N)	59.1, 59.2, 59.3, 59.4, 59.5, 59.6, 59.7, 59.8, 59.9, 60.0, 60.1, 60.2, 60.3, 60.4, 60.5, 60.6, 60.7
21 depth (m)	-155.9, -125.9, -95.9, -77.9, -71.9, -65.9, -59.9, -53.9, -47.9, -41.9,-35.9, -29.9, -23.9, -19.25, 15.95, -12.65, -10, -8, -6, -4, -1.5
124 times(UTC)	141955200, 141976800, 141998400, …, 144568800, 144590400, 14461200

Several Matlab programs are used to analyze the raw data, here is one of the program with user interface shown in figure 22, the program plots the temperature and salinity curves and calculates water density from temperature, salinity and pressure.

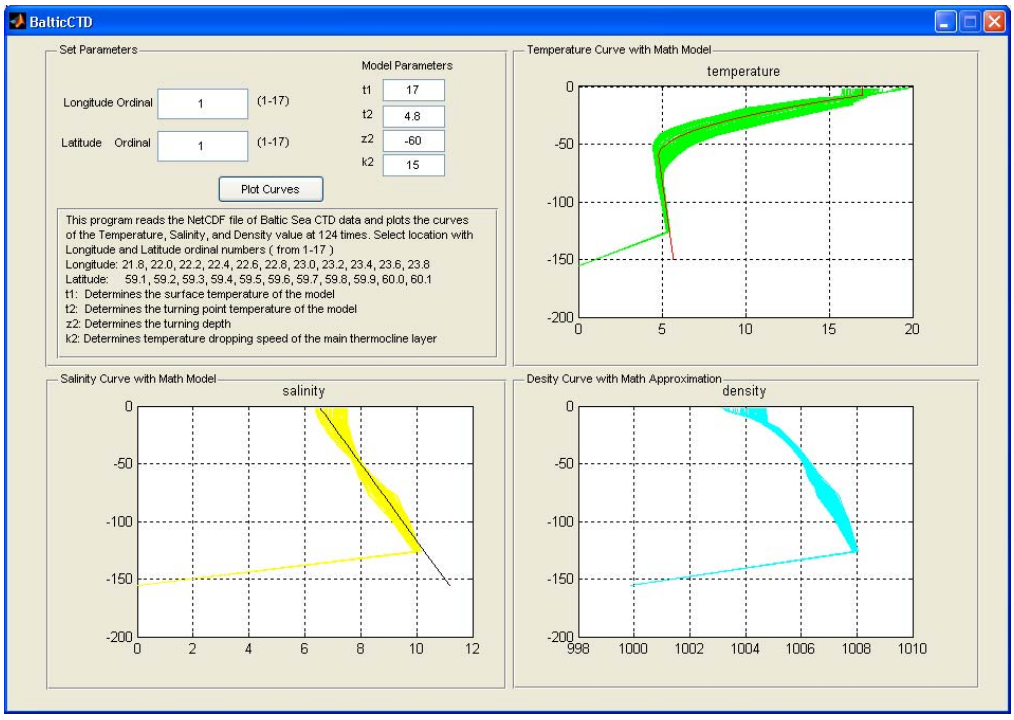

Fig. 22. Matlab software for analysis the raw data

In the first subsection, the user can set parameters such as longitude and latitude, as well as the approximate mathematical model parameters. The longitude and latitude ordinal

numbers are selected to display water properties from different locations, the temperature curves are shown in the second subsection, salinity curves are shown in the third section, and the calculated density curves are shown in the fourth section. The math model parameters determine a series of equations to approximate the temperature and salinity, for example, t1 determines the surface temperature, t2 determines the lowest temperature of the model, z2 determines the depth of the turning point (the deepest depth of the thermocline layer), and k2 determines temperature dropping speed in the thermocline layer. These will be defined and discussed more clearly later.

4.1 Temperature and salinity of Baltic Sea

4.1.1 Temperature profile of Baltic Sea in August

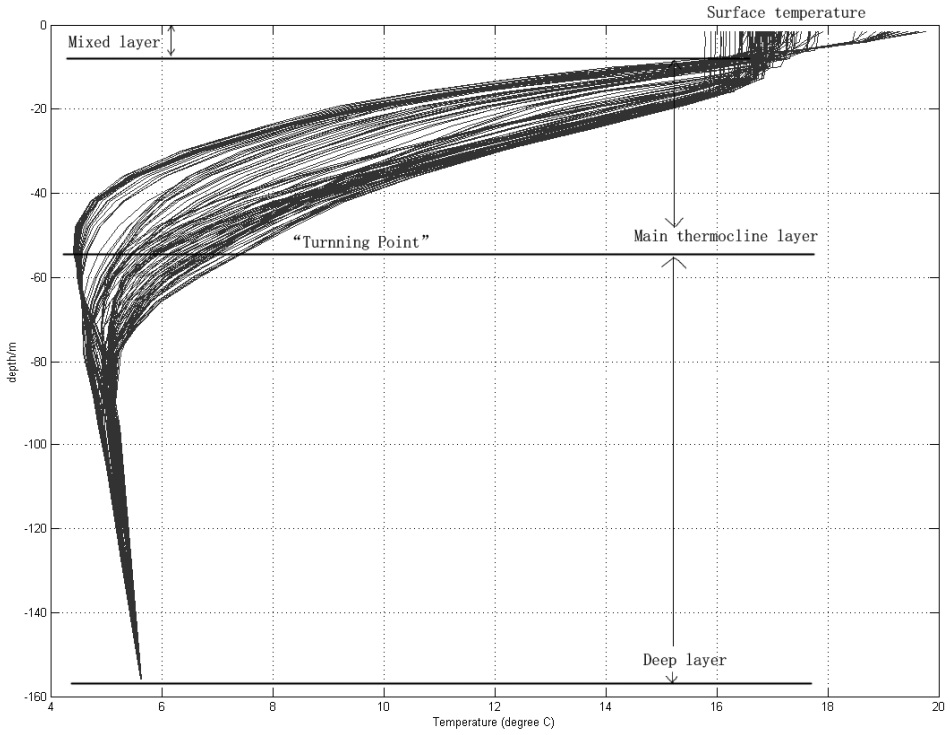

Fig. 23. Temperature curves in one location at 124 times from August 2008

In figure 23 are the temperature curves along with the depth in one sample location. From the curves, the temperature profile can be roughly divided into three layers:

Within about 15 meters from the surface, the temperature is invariable or drops slightly, that may be because the surface flow is very active and the water is mixed very fast, this is called the “mixed layer”. In August, the mixed layer temperature of Baltic Sea is from about 16 to 20 degree and it drops with time in August, but the thickness of this layer increases because of the cumulating and mixing of sunshine heat.

From 10 meters to 60-80 meters, the temperature drops rapidly in an exponential curve, this is called the “main thermocline layer”, the temperature drops due to the attenuation of the sunshine.

When the temperature drops to the lowest value at around 50-80 meters, which the author defines as the “turning point” of the temperature profile, the temperature stops dropping and goes up slightly, but can also be considered to be invariable around 5 degree, and this layer is called the “deep layer”.

Now the analysis of the temperature goes into two cases: temperature in one location at different times and temperature at one time in different locations in order to find out how the temperature changes with time and locations.
Tables below show some sample data:

Table 3. Temperature in one location at different times (sample)

Surface Temperature	17	19	17.5	17.5	17.8	17	16.8	17	16.9	16.5	15.8
Turning Point Depth	75	55	60	65	71	78	75	77	76	78	71
Turning Point Temperat	5	4.5	4.6	4.7	4.7	4.8	4.9	5	5	5	5

Table 4. Temperature at one time in different locations (sample)

Surface Temperature	19.06	19.04	19.02	19	18.99	18.97	18.98	18.96	18.97	18.93	18.9
Turning Point Depth	53.9	53.9	53.9	53.9	53.9	53.9	53.9	53.9	53.9	53.9	59.9
Turning Point Temperature	4.458	4.427	4.417	4.39	4.386	4.36	4.363	4.339	4.351	4.328	4.349

From table 3 and table 4 we can figure out some rules of the seawater temperature:
The seawater temperature does not change very much with locations in this area, but varies more with time. That means the temperature does not depend on location very

much (in a limited area), which is very good for us to estimate the temperature at different depths without considering the location.

Using Matlab, the water temperature can be modeled as three parts:

$$\begin{cases} T=T_0;\ (0<z<z_1,\ \text{the mixed layer}) \\ T=((depth+z1)/k_2).*((depth+z_1)/k_2)+t_1;\ (z_1<z<z_2,\ \text{the thermocline layer}) \\ T=T_1;\ (z>z_2,\ \text{the deep layer}) \end{cases}$$

4.1.2 Salinity profile of Baltic Sea in August

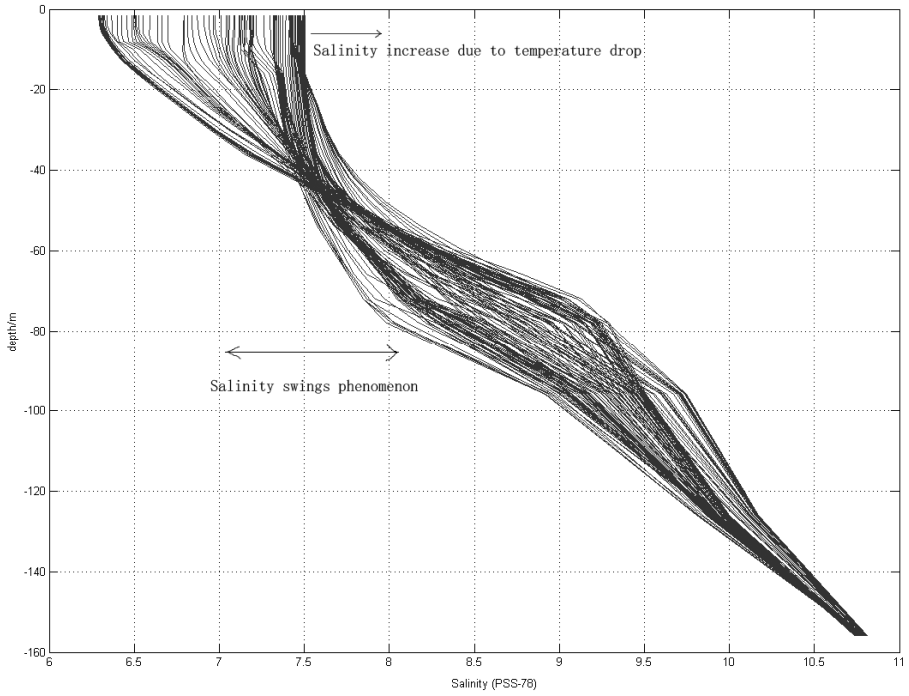

Fig. 24. Salinity profile plotted by Matlab from raw measurement data.

Figure 24 shows the salinity curves along depth from 124 times measurements in August 2008. From the curves we can conclude the properties of salinity in August:

From sea surface to about 40 meters, salinity increases with time. According to the discussion of temperature profile, the salinity increases due to the temperature drop. Seawater salinity has a changeful phenomenon from 50 meters to 80 meters depth, where the salinity changes rapidly. Figure 24 correspond directly with the research result of Finnish Environment institute: “Salinity changes abruptly in the depth of 50-80m

from the fresher surface waters to the saltier deepwater. This depth of rapid change in salinity is called a halocline and it is a permanent phenomenon in the Baltic."[27]

4.2 Water density of Baltic from raw data

The SWARM float controls its depth by buoyancy. Hence, water density is the most relevant physical quantity for the diving control. Density cannot be measured from the sensors directly but can be calculated from the measurement data. This part of this thesis will deal with the density calculation from the raw data using Matlab, and try to find out the properties of the seawater density.

The density (D) is calculated by the salinity (S), temperature (T), and pressure (P) using the UNESCO international equation of state for seawater [UNESCO Technical Papers in Marine Science, Vol. 44], Salinity (PSS-78), temperature (ITS-90), gauge pressure (millibar), and density (kg/m^3):[28]

```
r0 = 999.842594 + T * ( 6.793952e-02 + T * ( -9.09529e-03 + T * ( 1.001685e-04 + T * ( -1.120083e-06 + T * 6.536332e-09 ) ) ) ) +       S * ( 0.824493 + T * ( -4.0899e-03 + T * ( 7.6438e-05 + T * ( -8.2467e-07 + T * 5.3875e-09 ) ) ) + sqrt( S ) * ( -5.72466e-03 + T * ( 1.0227e-04 + T * -1.6546e-06 ) ) + S * 4.8314e-04 );

K = 19652.21 + 148.4206 * T - 2.327105 * T^2 + 1.360477e-2 * T^3 - 5.155288e-5 * T^4 + S * ( 54.6746 - 0.603459 * T + 1.09987e-2 * T^2 - 6.1670e-5 * T^3) – sqrt( S^3 ) * ( 7.944e-2 + 1.6483e-2 * T - 5.3009e-4 * T^2) + P * (3.239908 + 1.43713e-3 * T + 1.16082e-4 * T^2 - 5.77905e-7 * T^3 + S * ( 2.2838e-3 - 1.0981e-5 * T - 1.6078e-6 * T^2 ) + sqrt( S^3 ) * 1.91075e-4 )    + P^2 * ( 8.50935e-5 - 6.12293e-6 * T + 5.2787e-8 * T^2 + S * (-9.9348e-7 + 2.0816e-8 * T + 9.1697e-10 * T^2 ) );

D = r0 / (1 - P / K);
```

Where S and T come from the raw data, and pressure (P) is calculated from depth using the formula of Leroy & Prathiot 1997: "Depth-pressure relationships in the oceans and seas with corrective terms for Baltic Sea", depth (Z) in meters, positive, and gauge pressure (P) in millibar:[28]

```
k = (9.819176693 – Z * 2e-5) / (9.80612 – Z * 2e-5);
h_45 = Z * (1.00818e-2 + Z * (2.465e-8 + Z * (-1.25e-13 + Z * 2.8e-19)));
P = 1e4 * h_45 * k - Z * 1.8e-4;
```

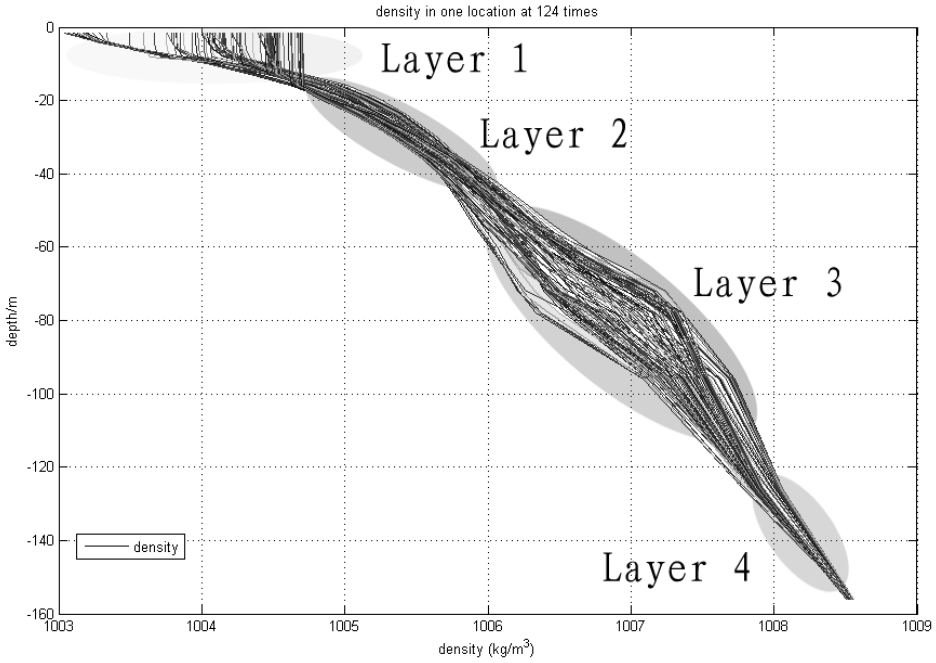

Fig. 25. Density curves of one location at 124 times

Figure 25 shows a sample of the density curves in one month at location 22.0E 59.2N, from the result we can see the density of Baltic seawater is nonlinear and time variant from the surface to the depth of 160 meters. There are mainly four layers identified in the figure:

Layer 1: from sea surface to about 15 meters depth, we can name it the “surface layer”, the density of seawater increases from about 1003 kg/m^3 to about 1004.7 kg/m^3 during 124 times (one month) measurements. Comparing the density curves with the curves of temperature we can conclude that the layer 1 water density increases with time due to the drop of surface water temperature, this can be visualized from the Matlab program “density_time_step.m” and “temperature_time_step.m” which give an animated display of the density and temperature data with time.

Layer 2: there is a layer from about 15 meters to about 40 meters, which is named the “static layer” by the author, in this layer, the density does not change very much with time, and the curves are quite uniform, it is good for us to predict the density of this layer although it is also nonlinear.

Layer 3: this layer is from about 40 meters to 100 meters depth, the author names it the “shaking layer”, it has significant oscillations of density especially around 80 meters.

Comparing this property to the salinity curves, we can conclude that the oscillations of water density in this layer are caused by the change of the salinity. The salinity data shows that at about 80 meters the salinity has the largest cyclical changes, this is why the density of the seawater changes a lot here, but the reason why the salinity changes so much at about 80 meters is unexplainable by the author using the raw data.

Layer 4: the deeper layer is from 100 meters to 160 meters, which can be called the "bottom layer", because it is going to reach to the deepest Baltic Sea bottom, in this layer, the density curves converge to a final density point, which means with increasing depth the less density changes with time. The change of the density in this layer is because the dense water is heavier and sinks to the deepest part of the ocean.

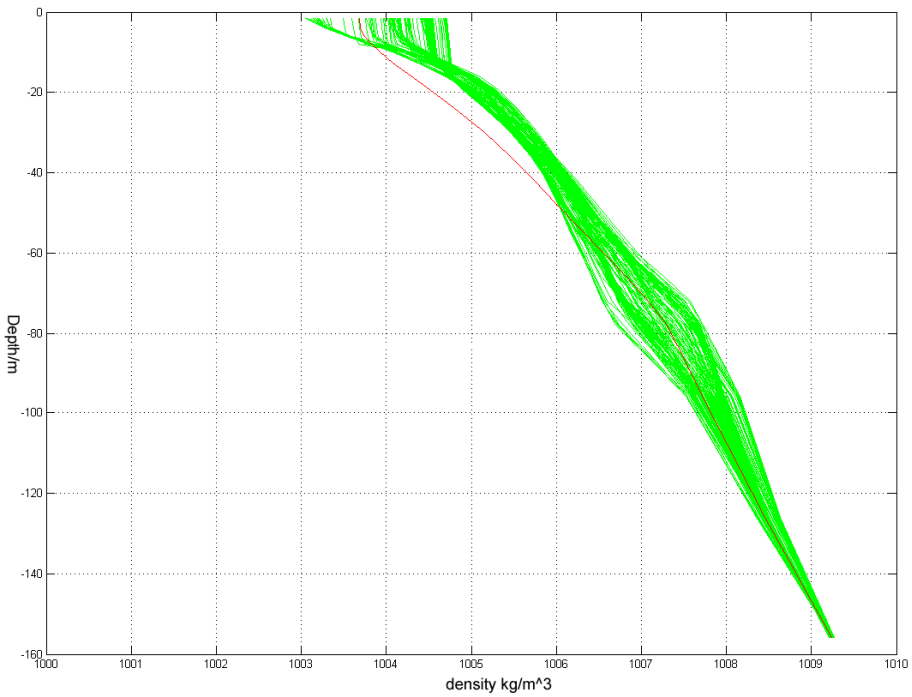

Fig. 26. Density curves and approximate polynomial

Figure 26 shows the density curves and the polynomial curve. The polyfitting does not correlate very well to the density curves because the polynomial fits to the mean of all the density data. In this figure, however, only one location (22.0E 59.2N) was given as a sample.

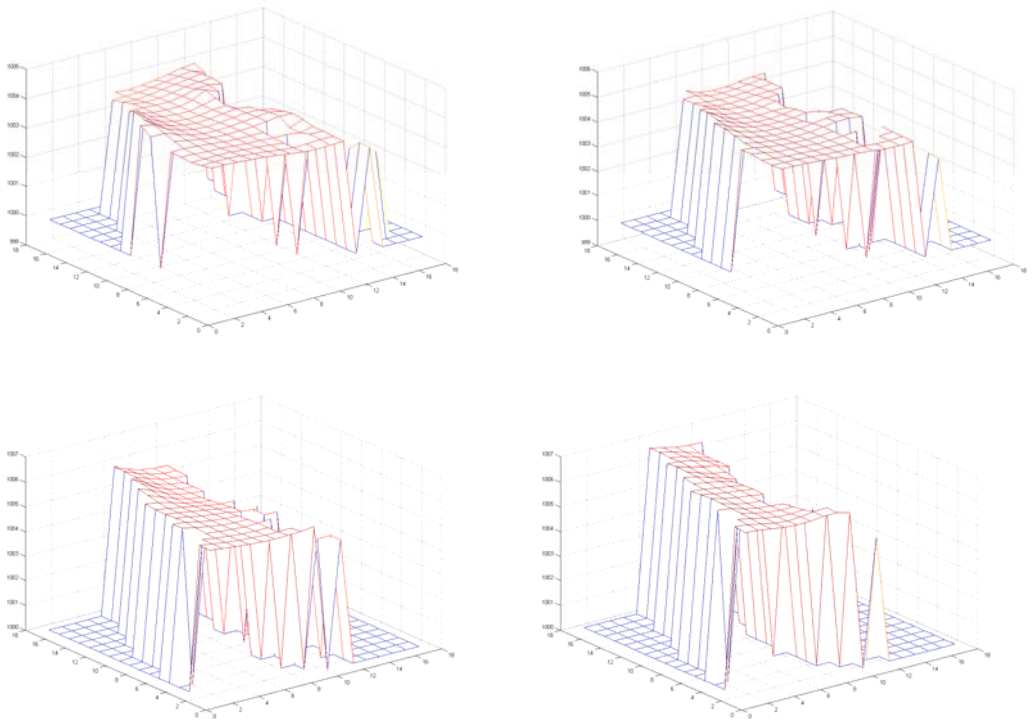

Fig. 27. Density graph in all locations in different depths

a b
c d
a) surface(1.5m) b)19.25m c) 41.9m d) 59.9m

Figure 27 shows the density transversal curve in different depths, figure a) is the density of the surface layer, which varies more than the other layers but still looks flat. The density in the surface layer changes around 1003.4 to 1003.8 kg/m^3, in the deeper layers of the Baltic Sea, water density changes much more slightly as figure b), c), d) shows. Note that the map scale of this graph is very large. Therefore, the real Baltic water density is almost invariant in a large geographical area, which makes it possible to model the seawater density without the longitude and latitude parameters.

Chapter 5

Diving control design

5.1 Dynamics of SWARM diver

5.1.1 Buoyancy

Buoyancy is a very important force for an underwater robot. It depends on the volume of the robot and the density of the seawater. An object which is immersed in a fluid is buoyed up by a force equal to the weight of the fluid displaced by the object. This is well known as Archimedes's principle. The buoyancy of a SWARM unit mentioned before is given by:

$-\rho(z,T,s)(V_f(z,T)+V_b)g$

$\rho(z,T,s)$: Water density

$V_f(z,T)$: Volume of the float without the piston

V_b: Volume of the piston out of the float. It is the volume of control.

5.1.2 Water drag

In the diving process, the robot suffers the drag of the water, which acts as resistance, even when the robot is freely floating, the drag cannot be avoided due to the current movement, and the robot in the water has to drift with the water.

The water drag depends on the water glutinosity and the surface of the robot. Generally, the drag in the diving process of the SWARM unit is:

$-\rho(z,T,s)\frac{SC}{2}\left|\dot{z}\right|\dot{z}$

$\rho(z,T,s)$: Water density

S: Surface of the basis of the float

C: Drag coefficient

5.1.3 Dynamics of the diver

According to the second Newton law, the dynamics of the diver are:

$$M\ddot{z} = (M - \rho(z,T,s)(V_f(z,T) + V_b))g - \rho(z,T,s)\frac{SC}{2}|\dot{z}|\dot{z}$$

The diving control of the SWARM float uses a piston engine which can change the volume of the float.

From the simulation, it can be seen that the maximum overshoot in the diving is very small (within 1 meter), which is because the diving velocity is very small and drag force is very large in water. This will be discussed more detailed in chapter 5.4.

5.2 Previous control and problems

A sliding mode controller is used in the previous diving control design, which worked satisfactorily for the dynamics control. However, the problem is that it needs a large number of piston movements, which is not economic for the energy budget. The control algorithm designed to follow a trajectory like the following figure shows:[29]

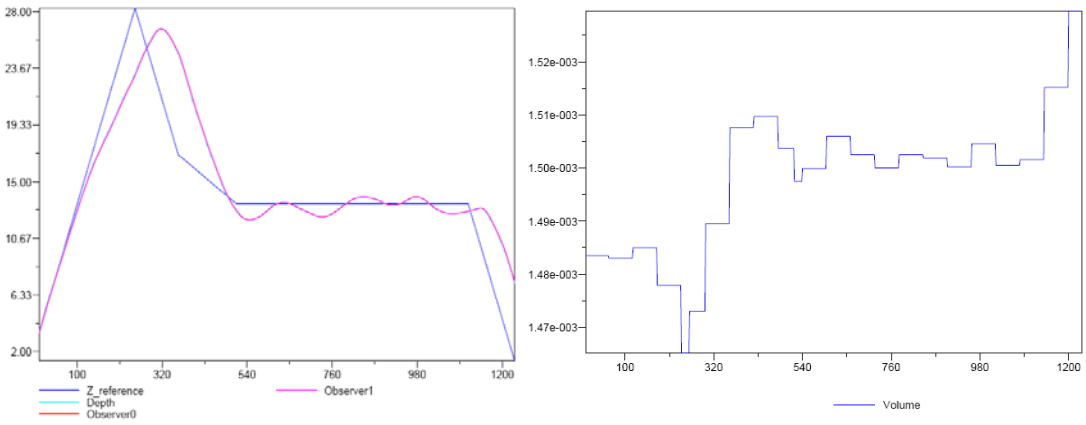

Fig. 28. Control using a Sliding Mode Controller (SMC)

Left of figure 28 shows a reference trajectory and test trajectory result, Right of figure 28 shows the piston volume changes.

The control algorithm controls not only the depth of the robot, but also the speed of the diving process. However, we are not considering how to follow a diving trajectory but how to dive to a certain depth directly with high accuracy and small overshoot. In other words, the previous control algorithm moves the piston all the time (one command every constant time) up and down to approach the designed trajectory which is not economic for the energy budget.

Energy consumption of SMC depends much on tuning of the controller and the diving reference trajectory. If the controller is tuned well and the reference trajectory is configured properly, the energy consumption of SMC can be reduced a lot. However, due to the nonlinear water density, it is not easy to make a perfect tuning and a best reference trajectory to optimize the energy consumption.

Major Problems:

1. Uneconomical energy consumption and computation. The sliding mode controller is an important control method used in the underwater vehicle depth control, but the floats of the SWARM system have low trajectory and dynamics requirement. As the mission needs the floats to dive to a certain depth or change from one depth to another, the sliding mode control is excess for the depth control in this mission.
2. Not reliable due to high dependency on sensor accuracy. The float has a CTD sensor, which can give a measurement every 1.4 second. The sliding mode controller needs to monitor the velocity of the robot and the water density, which are not reliable from the CTD sensor. Errors of the sensor data may cause fatal failure of the diving control.

The diving method in this thesis research is going to improve the control algorithm which moves the piston only when it needs to be moved. The goal is how to dive to a depth without large overshoot and error using less energy, but the trajectory in the diving is not so important. Thus we do not need to adjust the piston periodically to follow the trajectory, but control the piston to match the target density, and in the stability phase, the piston need to be moved only when the density changes— This is “Environment Based Control”.

5.3 New control concept

To control the dynamics of the float, the classical controllers need to compute and adjust the piston all the time during the diving process which costs much energy consumption. The new control algorithm will not use any of the classical control algorithms. As discussed in last chapter, water density does not change very much in one month and in a limited area, which enables the possibility to estimate water density for a certain depth. The new control method uses a density model built from previous data to reduce the piston movements in the diving process.

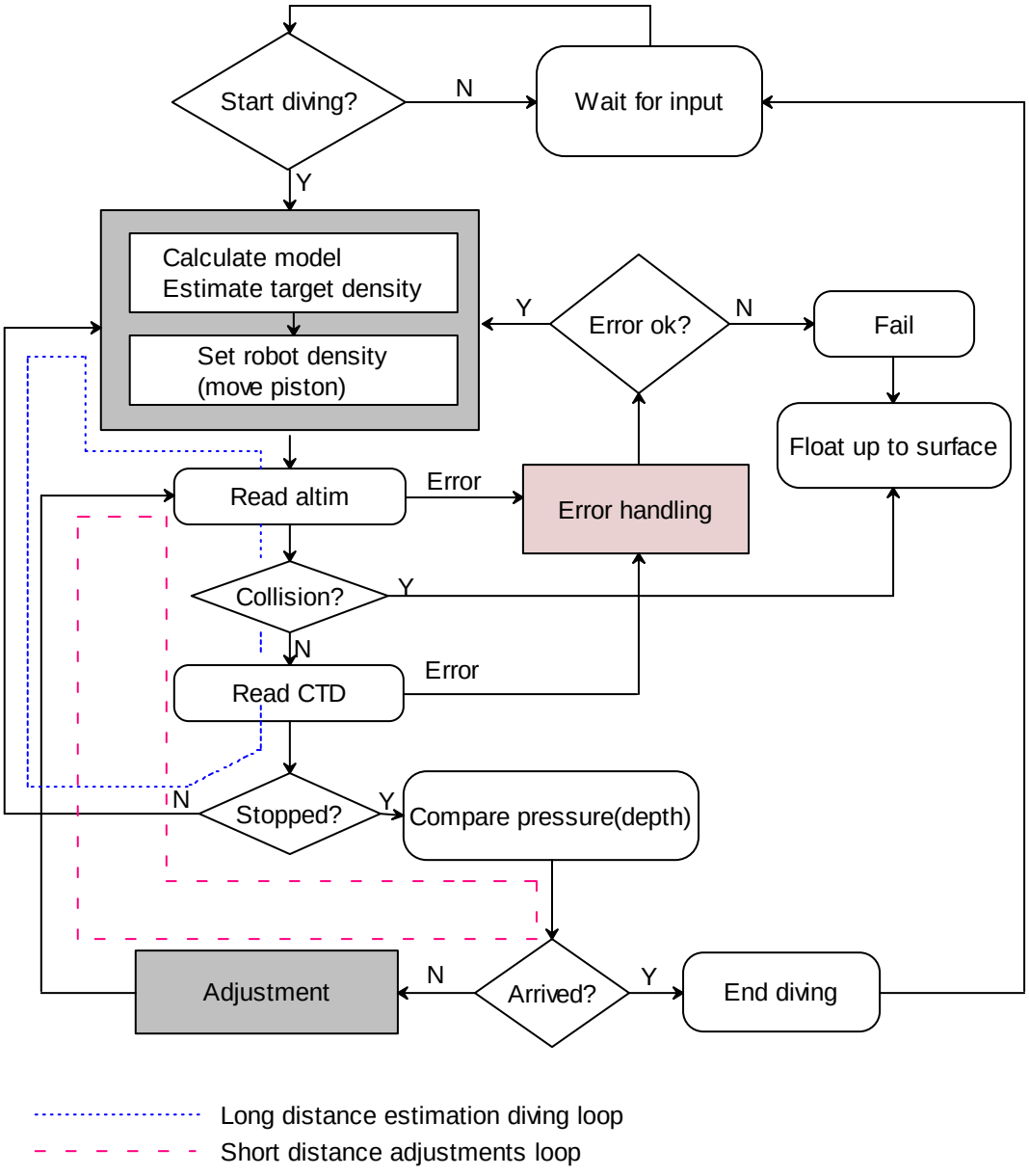

Fig. 29. Environment based diving control algorithm

The environment based control algorithm has two sections: one long distance estimation diving and several short distance adjustments as figure 29 shows.

5.3.1 Long distance estimation diving

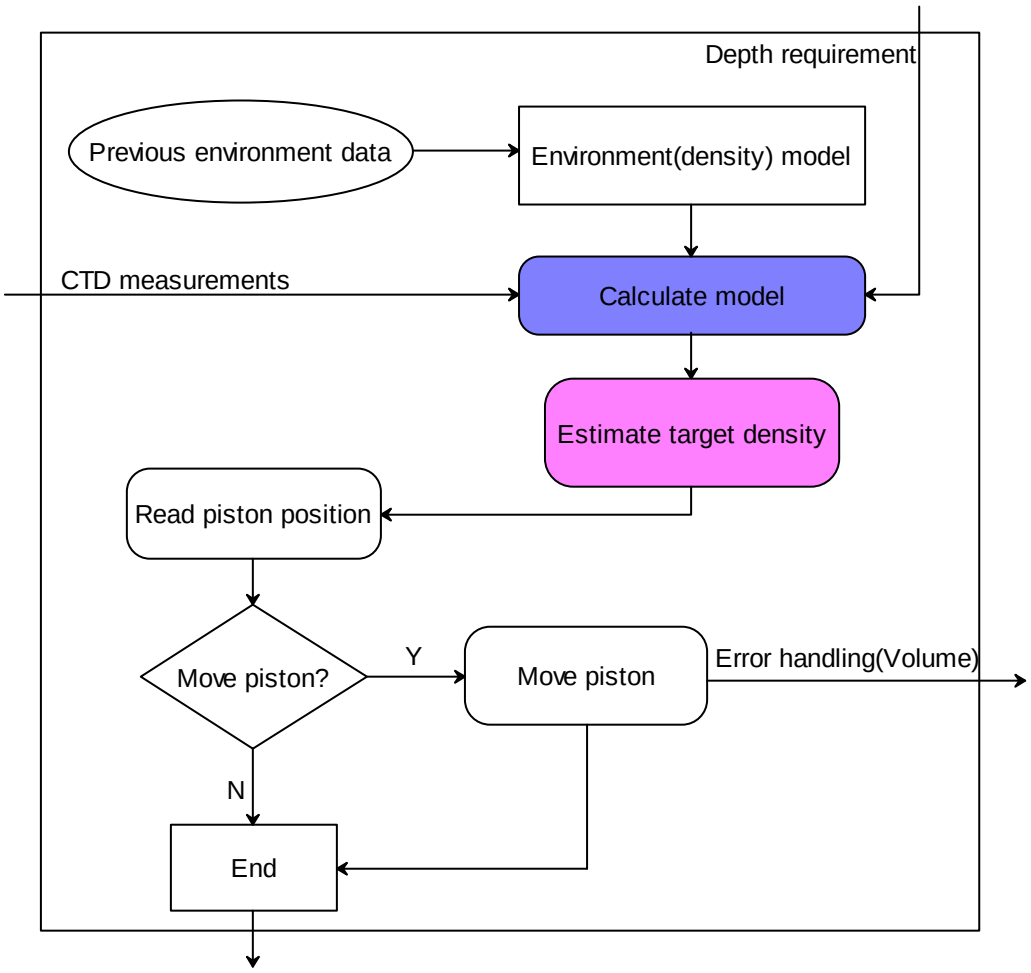

Fig. 30. Calculation algorithm of environment model

The long distance estimation diving uses the density model to estimate the water density of target depth. As discussed in last chapter, the seawater density does not change very much in one month and in a limited area of Baltic Sea, so that we can estimate the water density of the target depth by the density model built from previous measurement data. The first model uses a polynomial to approximate the mean density of the 752556 calculations from the raw data. The surface layer cannot be effectively modeled by this polynomial, thus the polyfitting is divided into two part, 0-6 meters and 6-160 meters.

$$\bar{D} = a_1 Z + a_0; \qquad (0<Z<6)$$

Here,

$$a_1 = -0.01458064, \quad a_0 = 1003.65416396$$

$$\bar{D} = a_7 Z^7 + a_6 Z^6 + a_5 Z^5 + a_4 Z^4 + a_3 Z^3 + a_2 Z^2 + a_1 Z + a_0; \qquad (6 \le Z<160)$$

Here,

$$a_7 = 5.82791451e\text{-}13, \; a_6 = 2.93618412e\text{-}10$$
$$a_5 = 5.80359192e\text{-}08, \; a_4 = 5.74515221e\text{-}06$$
$$a_3 = 3.01506402e\text{-}04, \; a_2 = 7.94346722e\text{-}03$$
$$a_1 = 3.38527732e\text{-}02, \; a_0 = 1003.71941686;$$

In the diving process, if the water density model has large error, we can update the model and make new estimation in the long distance diving process as figure 30 shows.

5.3.2 Short distance adjustments

The short distance adjustments are used to correct the error by long distance estimation diving. When the robot does not achieve the target depth within satisfying error, one or more adjustments are made to approach the target depth. Assume $k = \Delta D / \Delta z$ is the slope coefficient of water density curve, where Δz is depth difference and ΔD is water density difference in depth interval Δz, in a short distance, the water density can be considered to be linear, so that if the robot needs to be adjusted a depth error of Δz, the robot should change density by $\Delta D = k\Delta z$.

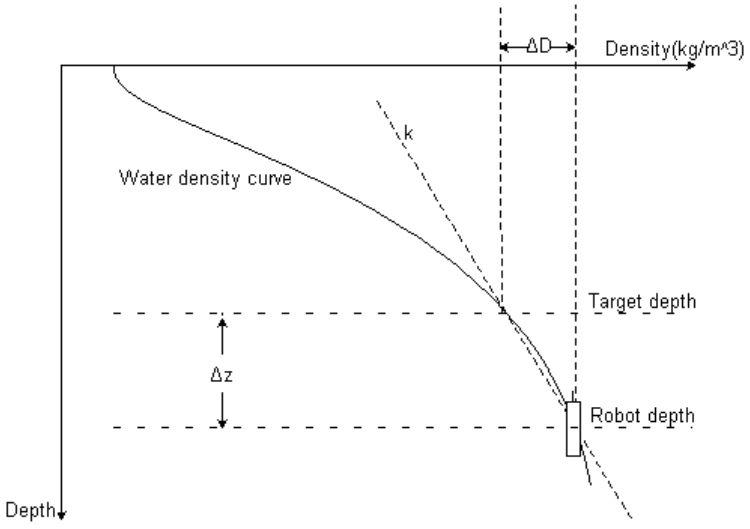

Fig. 31. Short distance adjustment algorithm

The first short distance adjustment uses the water density slope coefficients estimated from the previous environment model shown in figure 32:

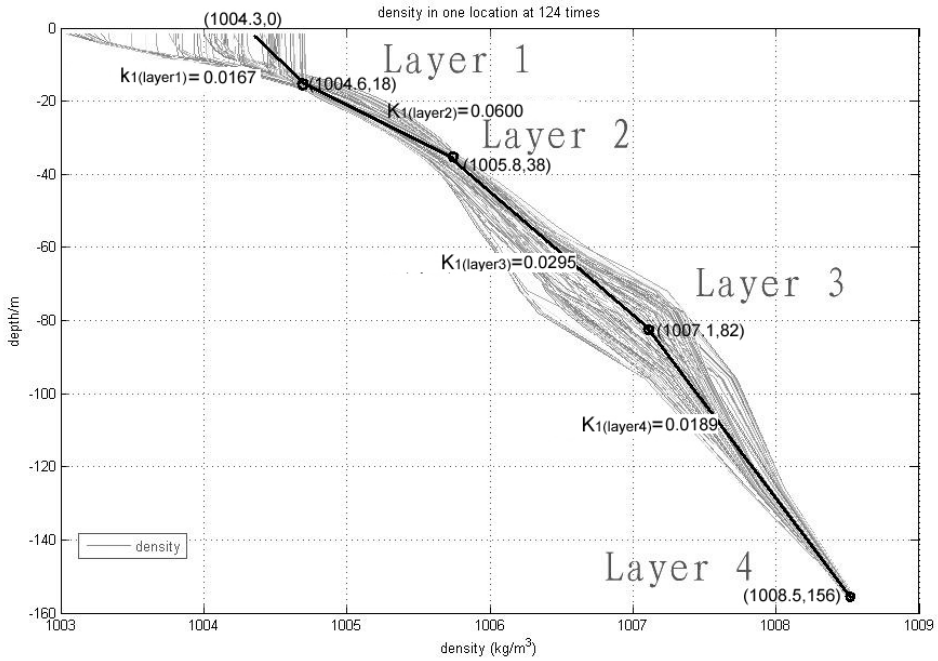

Fig. 32. Slope coefficients of density curve in different layers

The estimated slope coefficients k_1 for the four layers water density are:

$$k_1 \doteq \begin{cases} \dfrac{1004.6-1004.3}{18-0}=0.0167 & (0<z\le 18) \\ \dfrac{1005.8-1004.6}{38-18}=0.0600 & (18<z\le 38) \\ \dfrac{1007.1-1005.8}{82-38}=0.0295 & (38<z\le 82) \\ \dfrac{1008.5-1007.1}{156-82}=0.0189 & (82<z<156) \end{cases}$$

If the robot has a depth error of E_1 after the long distance diving, the estimated robot density of adjustment should be $\Delta D_1 = k_1 E_1$.

From the first adjustment, we can get real water density slope coefficient at the robot depth using the adjusted density and depth: $k_2 = \dfrac{\Delta D_1}{\Delta z} = \dfrac{\Delta D_1}{E_2 - E_1}$. Sequentially,

$k_i = (\frac{\Delta D_{i-1}}{E_i - E_{i-1}})$ is the density slope coefficient at i th depth. Therefore, the i th adjustment density of the float is determined by: $\Delta D_i = (\frac{\Delta D_{i-1}}{E_i - E_{i-1}}) \cdot E_i$

Where E_i is the ith depth error, and ΔD_i is the ith adjustment density, (i>1)

Figure 33 shows an example of two times adjustment to achieve the target depth, which gives a clear explanation of how the adjustment algorithm works.

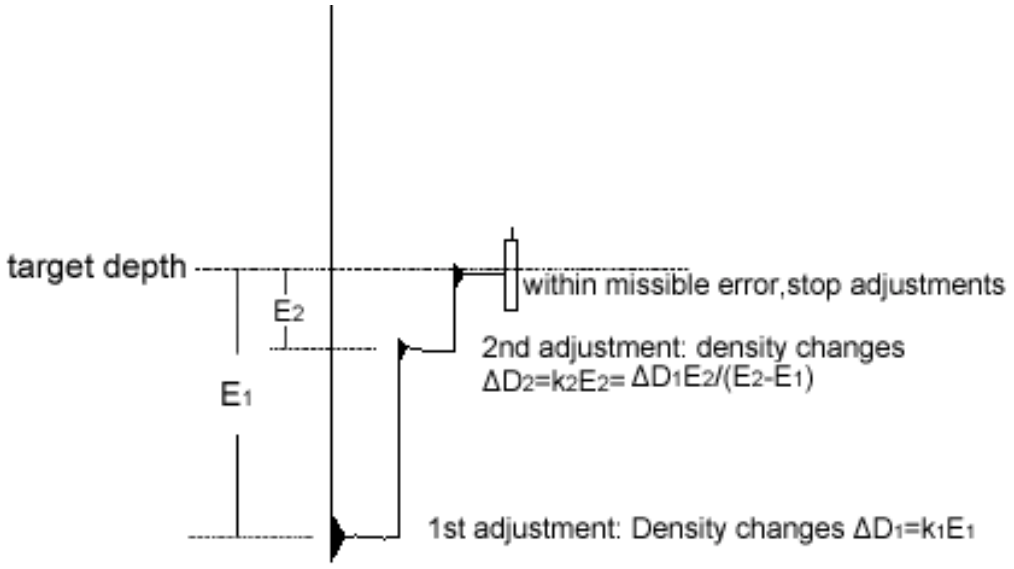

Fig. 33. Example of two times adjustment

The first adjustment uses a density slope coefficient k_1 which is estimated from previous data shown in figure 32, the second slope coefficient is $k_2 = \Delta D_1 / (E_2 - E_1)$ which is determined by the first adjustment. In the example of figure 33, after two adjustments, the robot arrives at a depth with satisfying error.

In the simulator test, the author added some intended error to the long distance estimation diving, even if the depth error is about 20 meters, the robot can adjust to target depth within 0.5 meter error by about 2 to 5 adjustments.

In the diving tests, the author found a phenomenon that the adjustments performances are different above the target depth and below target depth, most time the upward adjustments density were smaller than the required density change, contrarily, larger in the downward adjustments. The reason is shown in the following figures:

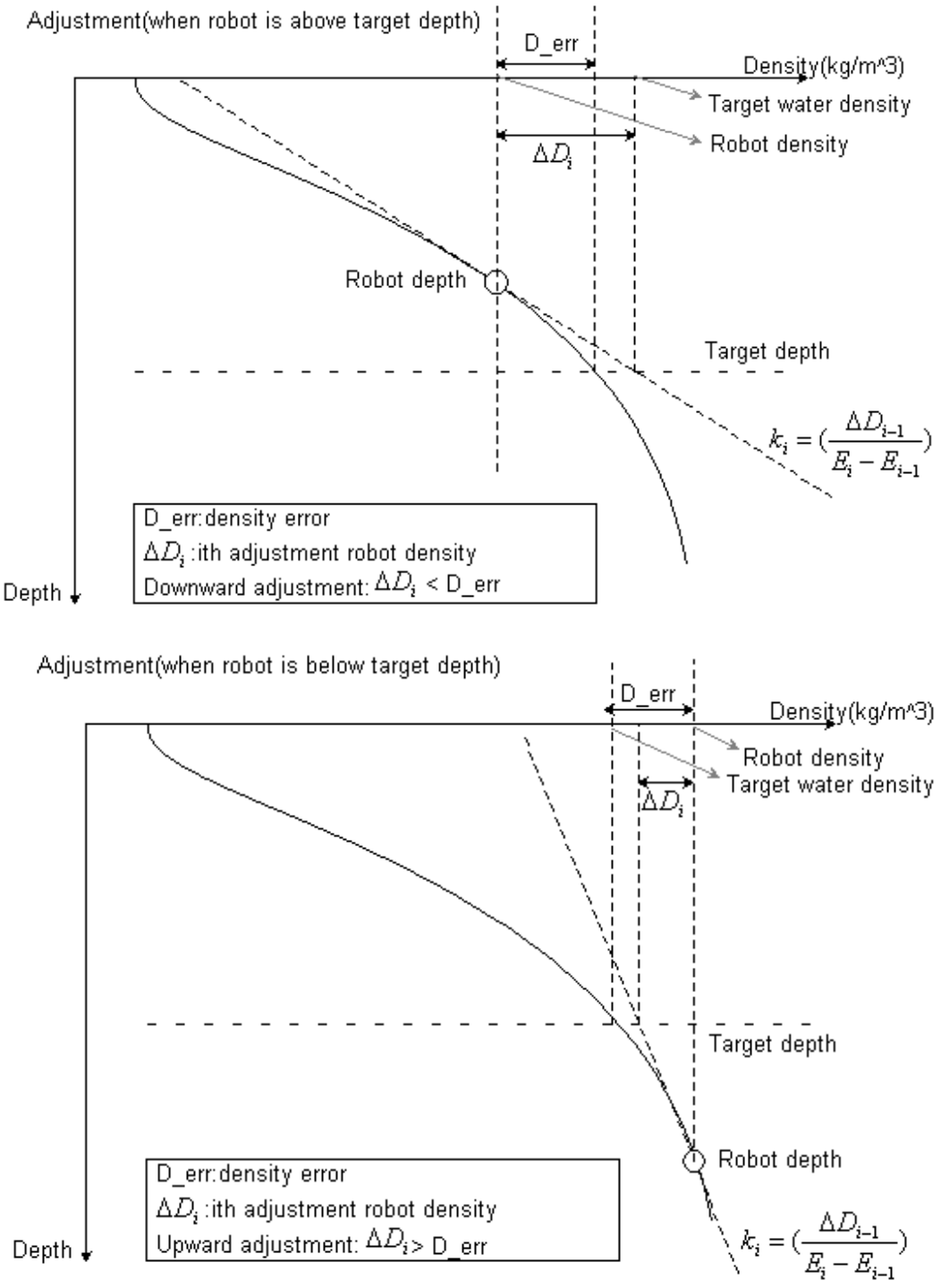

Fig. 34. Upward adjustment and downward adjustment

5.4 Evaluation of environment based control algorithm

Advantage:

Energy consumption is improved very much. The energy consumption highly depends on how often the float moves the piston, the static friction and the pressure is large, so that the frequency to move the piston should be as few as possible. In this control algorithm, the float has already had a model of the density profile with depth, the long distance estimation diving makes a big step to approach the target depth by only one movement, and a few more adjustments (piston movements) are made to get to the target depth within satisfying error. The smaller error requirement, the more adjustments needed. In the simulation test, the robot can dive to target depth using less than five piston movements in most of the cases.

The new control algorithm is simple and easy to handle. The diving control uses the experiential data and model, which means before the diving, the robot already knows something about the environments, the measurement and computation error will not cause fatal error.

Disadvantage:
Highly depends on environment data in hand, lots of study needed to model the environment. To build a model, a large number of measurements are needed, and a lot of study needed to find out the rules of the environments.
The environment based control is inflexible to be implemented in different environments. It is the main limitation of this control concept. For instance, the data comes from August, but it may be quite different with the real environments state of May, all the information and control need to be modified in order to fit the real environments. And it cannot be easily used in other oceans and fresh water because we don’t know whether the water density changes not too much with time and location. This method is only design for the SWARM project. However, similar concept can be used in other underwater robot control.

5.5 The effect of drag force coefficient

This section discusses about the effect of the drag coefficient. Damping is one of the key influence factors in control theory, which has either positive or negative effect to the system. Anyhow, damping cannot be avoided in most of the control system. In the SWARM diving control system, damping comes from the liquid drag force, which

depends on three parameters: the vertical drag coefficient C, the horizontal cross-sectional area S and the vertical velocity $\dot{z}$ of the float:

$$-\rho(z,T,s)\frac{SC}{2}\left|\dot{z}\right|\dot{z}$$

The state space of the float is:

$$\begin{bmatrix} z \\ \dot{z} \\ \ddot{z} \end{bmatrix}$$

$$\begin{cases} \ddot{z} = g - \dfrac{\rho(z,T,s)(V_f(z,T)+V_b)g}{M} - \dfrac{\rho(z,T,s)\dfrac{SC}{2}\left|\dot{z}\right|\dot{z}}{M} \\ \dot{z} = \int \ddot{z}dt \\ z = \iint \ddot{z}dt \end{cases}$$

As water density $\rho(z,T,s)$ is a nonlinear function of depth z, and drag force $\rho(z,T,s)\frac{SC}{2}\left|\dot{z}\right|\dot{z}$ is a nonlinear function of water density and velocity of the float, this system cannot be easily solved. In the simulator, RK4 scheme (a fourth order Runge-Kutta integration method) is used to approach this nonlinear system: In a small depth interval Δz, the water density is assumed to be linear, and the next density is determined by the present density plus the product of the size of the interval Δz and an estimated slope. The slope is a weighted average of slopes:

k_1 is the slope at the beginning of the interval;

k_2 is the slope at the midpoint of the interval, using slope k_1 to determine the value of water density at the point $z + \Delta z / 2$ using Euler's method;

k_3 is again the slope at the midpoint, but now using the slope k_2 to determine the density value;

k_4 is the slope at the end of the interval, with its density determined using k_3.

The program calculates the velocity and acceleration of the float and at $t+\Delta t$ the program updates the water density from the environment, and calculates the dynamics

of the float again. When the time interval is small enough, the simulator can approach to the nonlinear system.

In the diving control test, the simulator picks up a random location and a random start time to test the diving algorithm in different conditions. To simulate the relationship between overshoot and drag coefficient, every diving test should be at the same location and time, therefore some small changes should be done with the simulator. The location is fixed in ‘drifter.cpp’, and time is fixed in ‘sssim-env.cpp’. The drag coefficient can be set in ‘drifter.h’. Figure 35 shows the diving test screenshots with different drag coefficients. In the simulation program, the density of the robot is set to be 1004 kg/cm^3, and the robot is dropped into the seawater and dives freely to about 11.3 meters. The default drag coefficient of the robot is 1.9 and drag coefficients from 0.0 to 3.0 are tested here and the diving track can be seen from the figures. As the simulator is not ideal model of the robot dynamics, when drag coefficient is 0, the robot also stops after a long time oscillations which should not be mathematically.

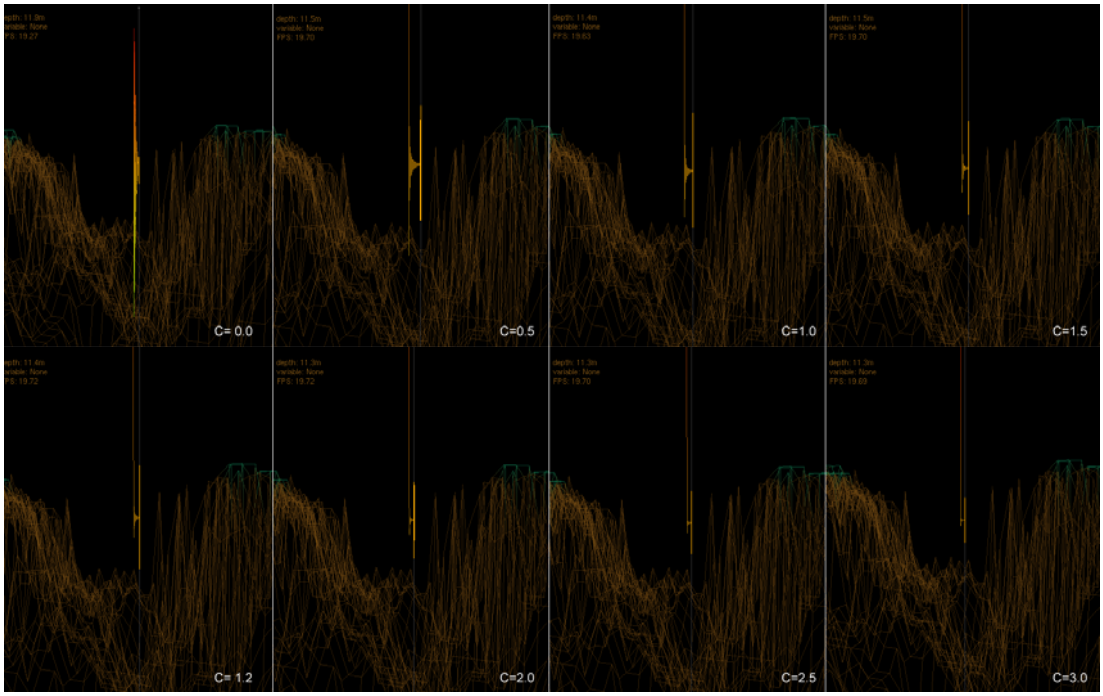

Fig. 35. Diving test with different drag coefficients

The overshoot, maximum velocity and settling time are the concerned features in the diving test, the simulation results of them are shown in table 5. In most of the control system, overshoot is considered to be a major nuisance. In the diving control of SWARM float, overshoot refers to the maximum depth exceeding the final depth.

Large overshoot may cause collision to the seabed, so that it should be avoided as much as possible. From the diving results in the simulator as figure 35 and table 5 shows, the overshoot is not very large when drag coefficient is larger than 0.5. The maximum velocity in the diving process determines the smallest measurement interval. As table 5 shows the maximum velocity with drag coefficient 1.9 is 0.0882 m/s, which is low enough for the robot to take CTD measurements. The settling time is shorter with larger drag coefficient, as C=1.9, the settling time to dive to 11.3 meters is about half an hour.

Table 5. Diving simulation result with different drag coefficients

drag coef / test result	0.0	0.1	0.2	0.3	0.4	0.5	1.0	1.5	1.9	2.0	2.5	3.0
max velocity (m/s)	0.3303	0.2521	0.2143	0.1891	0.1715	0.1588	0.1185	0.0983	0.0882	0.0857	0.0782	0.0706
overshoot (m)	11.4722	5.9199	3.8525	2.7835	2.1481	1.9118	1.0647	0.7823	0.6613	0.6411	0.5604	0.4999
settling time (s)	7044	4619	3650	3001	2902	2878	2418	2266	2008	2001	1933	1703

The following figure 36 and 37 show the overshoot and maximum velocity with different coefficients, from the plot we can see the curve is flat from c=1.0 to c=3.0, that means the overshoot and maximum velocity do not change very much when drag coefficient is larger than 1.0, so that small error of the drag coefficient around 1.9 will not cause problems of overshoot and diving velocity.

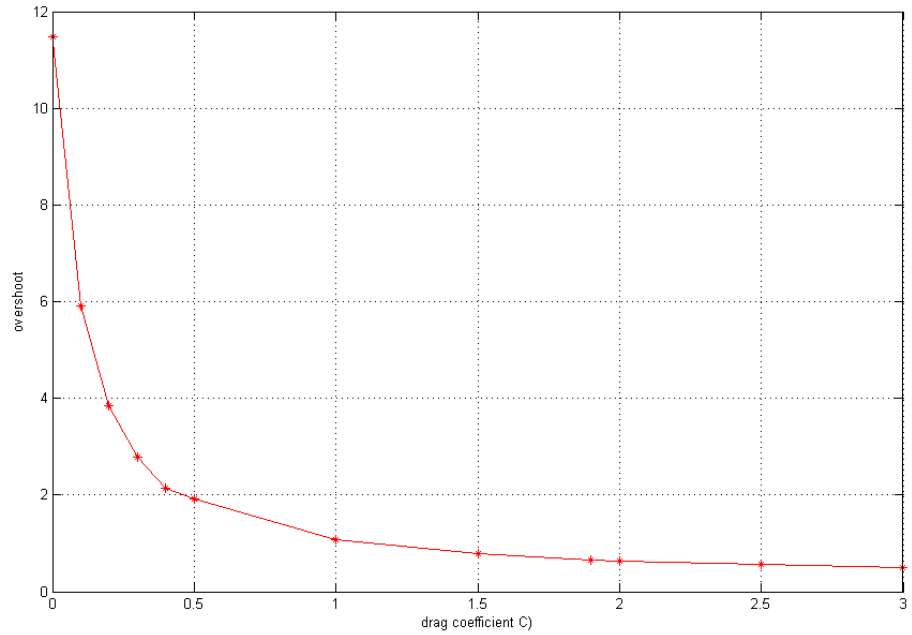

Fig. 36. Overshoot with different drag coefficients

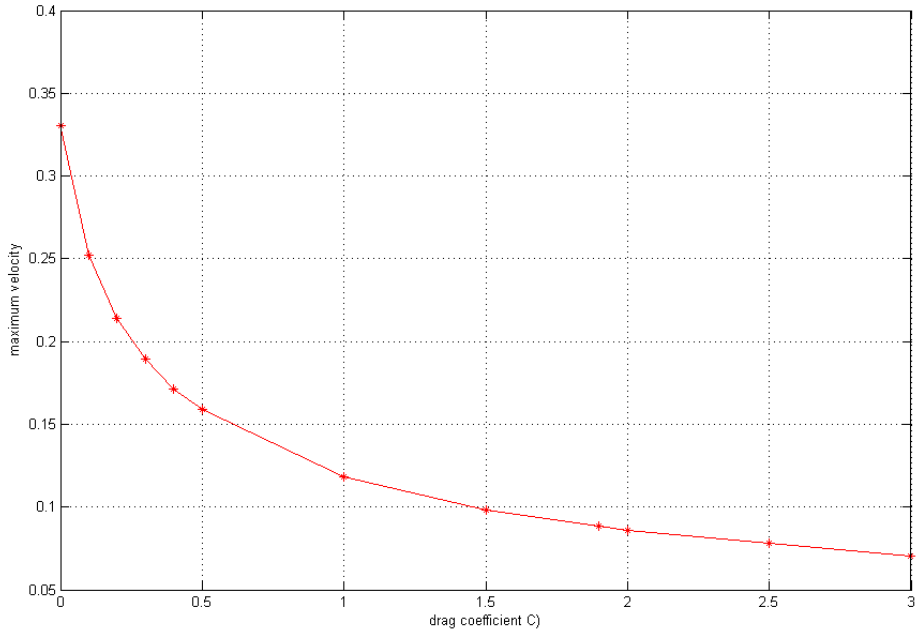

Fig. 37. Maximum velocity with different drag coefficients

From the above discussions, we can conclude that larger drag coefficient is better for the diving performance, which has smaller overshoot, lower diving velocity and shorter settling time. Thus, to improve the diving control, it is better to increase the drag coefficient. One way is to make a flat ring around the robot which can increase

the cross-sectional area. For instance, Argo float has this kind of design as figure 38 shows:

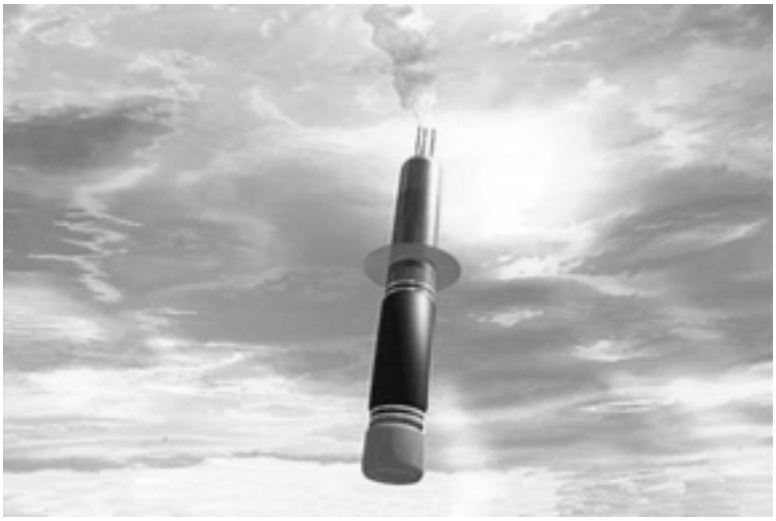

Fig. 38. Argo float with stability ring

The velocity and acceleration are small is not only because high damping, another reason is that the water density changes very slightly, and the robot density is very close to the water density, so that the gravity force and buoyancy are very close, the following calculations show the forces on the robot as example:

Assume that the mass of the robot is 42kg (in ENSIETA report), surface water density is 1003.5kg/m^3, and if we set the robot density to 1004kg/m^3, the forces of the robot in the surface water are:

Gravity force:

$$G = mg = 42kg \times 9.80665 = 411.8793N$$

Buoyancy:

$$F_{buoyancy} = \rho_w V_r g = \rho_w \frac{M}{\rho_r} g = 1003.5kg/m^3 \times \frac{42kg}{1004kg/m^3} \times 9.80665 = 411.6742N$$

Join force of gravity and buoyancy is:

$$G - F_{buoyancy} = 411.8793 - 411.6742 = 0.2051N$$

Thus, the robot is almost neutral buoyancy, if we take account of the drag force, join force on the robot is smaller, and this is why the diving velocity, acceleration and overshoot are such small in the diving process. This enable the implementation of environment based diving control, which matches the robot density with water density and dives directly to the target depth without large overshoot.

5.6 Diving test and result in the simulator

Figure 39 shows a diving test without short distance adjustments, this happens when the density model is very good and the water density estimation is very close to the real value. The robot dives freely by the forces of gravity, buoyancy and drag force. Once the density is configured, there is no more piston movement during the diving process. An exact density model has great advantage to reduce energy consumption. Figure 40 is the diving trajectory of the diving test, the diving takes about 6 minutes to arrive at 25 meters depth. From the figures, it is easy to see the overshoot and error are very small.

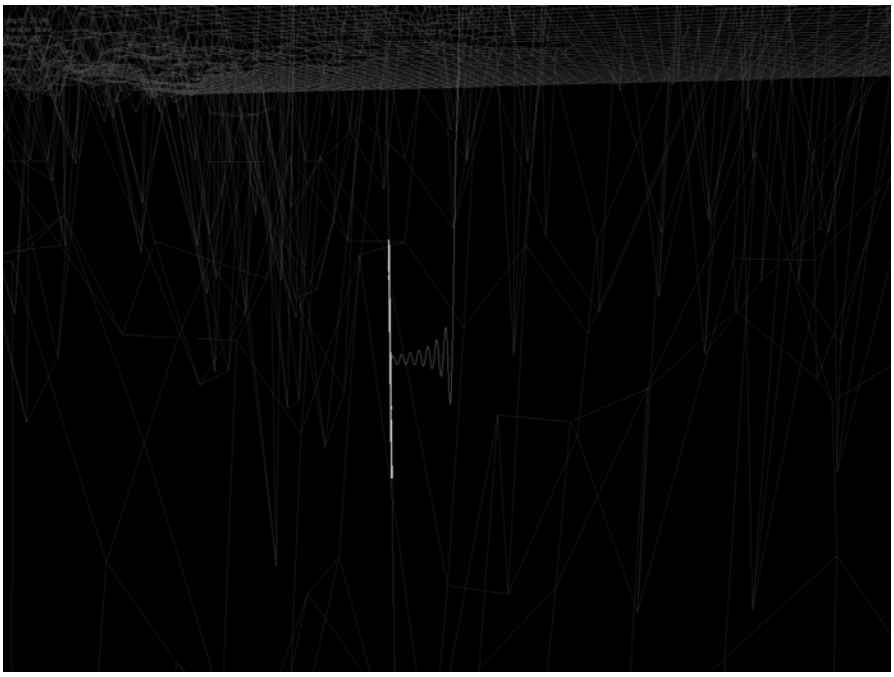

Fig. 39. Arrives target density in simulator.

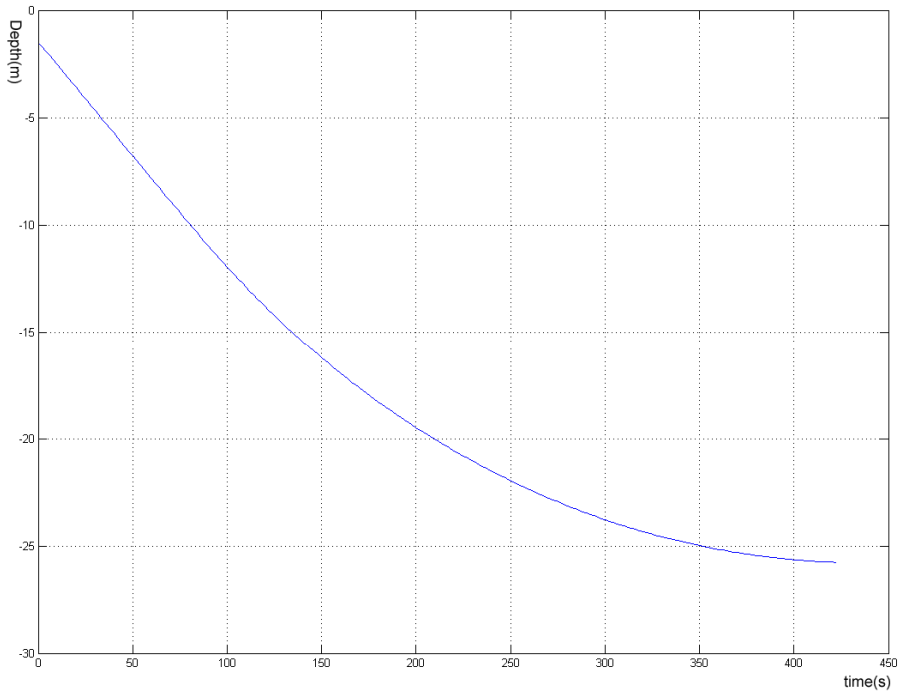

Fig. 40. Depth vs. time curve

The diving test in the simulator has very good performance, which is because the environment model uses the same data as the simulator. The model and density estimation cannot always be exact with the real water density, therefore short distance adjustments are needed for most of the actual cases.

In the beginning, the short distance adjustment algorithm is not mature, it just adds or reduces a constant small density every step to approach target depth, which is shown in figure 41.

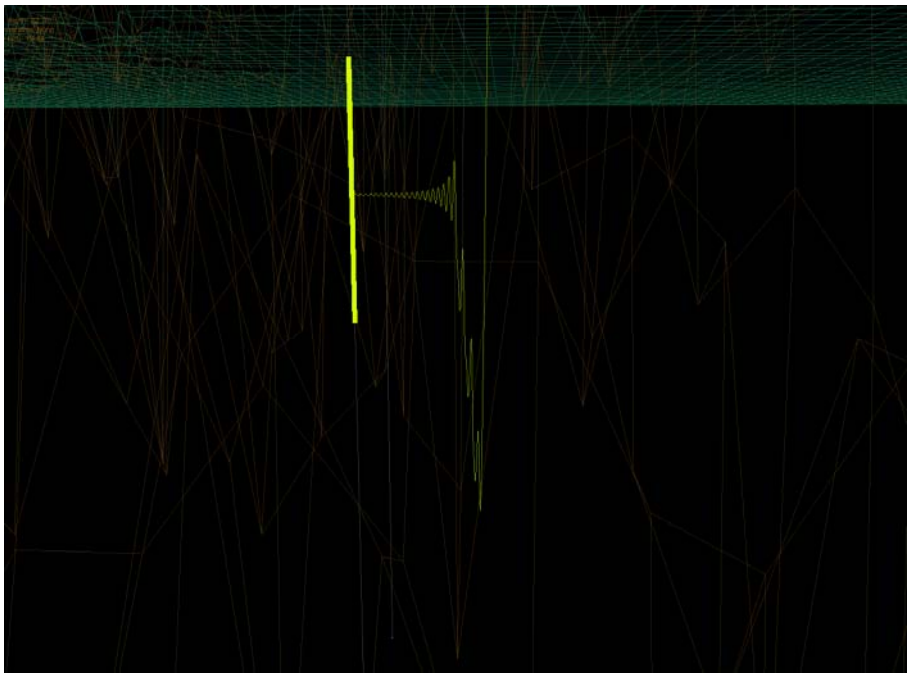

Fig. 41. One diving test with three simple adjustments

The robot dives to the predicted depth which is about 2 meters further than the target depth, and then it compares the pressure, makes three adjustments to go back to the target depth.

Problems to be solved:

1. The prediction error here acts as an overshoot which we don't want it happens. Never go further because it may not be safe (collision). So the overshoot should be adjusted before it happens. Correct the estimation beforehand: use the measurements near and before the target depth to do a second prediction and check if the previous prediction has large error, if does, move the piston once more and correct the final depth.

2. The adjustments add or reduce the density by 0.1 kg/m^3 every step (near the target area it is about 0.5 meter depth) to approach the target, as the figure above shows, it adjusts 3 steps to arrive the target, but we would like the adjustments as few as possible. Assume that the prediction deviation is very large, for example 10 meters, then it need about 20 adjustments and a long time to correct the depth error. And also the adjustments will cost large energy consumption. The adjustment density should be determined by the error, measurements and the model to reduce the steps of adjustments but not simply add or reduce a constant value.

Thus, the new adjustment algorithm discussed in 5.3.2 is used for the depth error adjustment. In the following diving tests, the author adds some intended error to the density estimation, and tests how the short distance adjustment works. Figure 42 shows one of the tests, the depth error of the first diving is about 7 meters and the robot adjusts to the target depth by only two piston movements.

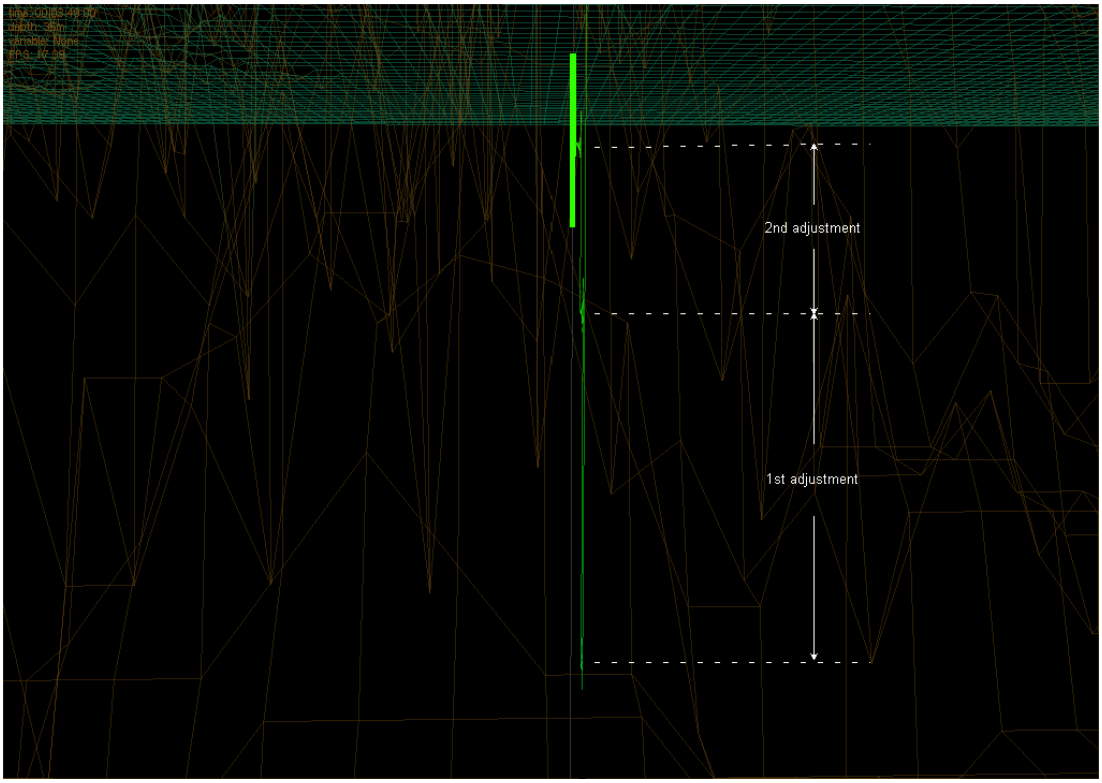

Fig. 42. Short distance adjustment test in simulator

In most of the cases, the robot needs only 2 to 5 adjustments to achieve target depth within 0.5 meter depth error.

Using the simple constant step adjustment, the diving control around the first density layer 0-18 meters is much more difficult. In the diving tests, when the target depth is within 10 meters, the program might goes into an endless loop in which the float goes up and down between surface and 10m, shown in figure 43.

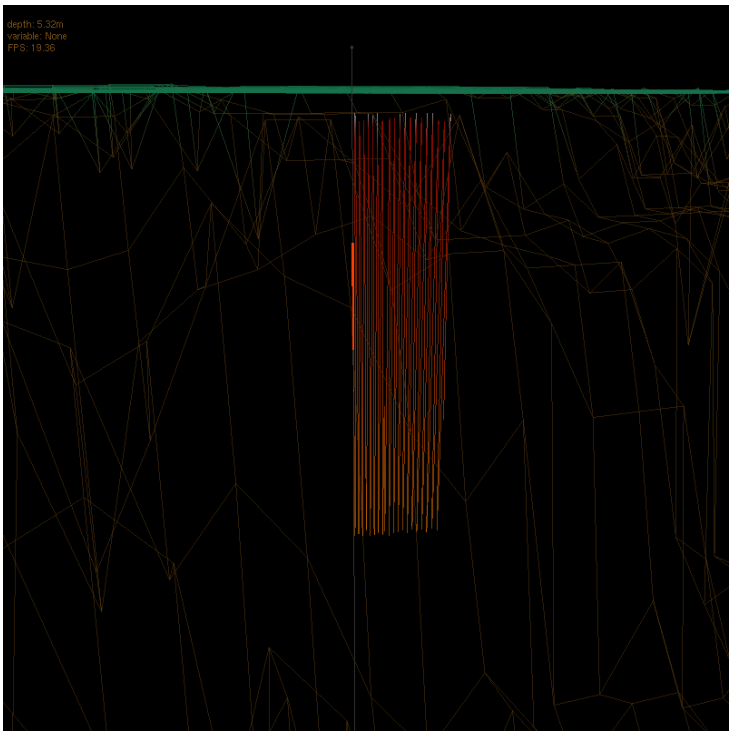

Fig. 43. Diving problem around small depth using simple step adjustment

This error is due to the water density changes very slightly from 0m to 10m, sometimes it is vertical invariable in this depth, the test shown in the figure is diving to 9 meters, the program try to get to 9 meters and keep the error within ±1 meter (8m-10m), but when it changes the density of the robot to approach the target depth, the float either goes up to the surface or goes to more than 10 meters, when it stopped at a depth of more than 10 meters, the program reads the pressure sensor and finds that the error is larger than 1 meter, then it will try to go up a little, however, because of the invariable density, it goes to the surface instead of 9 meters and then it dives down again and floats up again…

The new short distance adjustments method found later by the author can solve this problem, because the adjustments take account of the slope of the water density, the endless adjustment is not a problem any more.

5.7 Measure water density from robot dynamics

Here comes a new question: how to measure the water density. Firstly, we can use the CTD measurements: conductivity, temperature and pressure to calculate the density.

But the problem is that the robot in TKK has a CTD sensor with low accuracy conductivity, which is said to be 20% error. So the conductivity error should be considered as a key limitation of the density calculation. However, from the dynamics equation of the robot we can see that the density can be calculated from the velocity and acceleration of the robot, and they can be calculated by the pressure measurement, that means we need only to use the pressure without the conductivity. Which method is better will be discussed in this part.

The first methods: using the CTD data to calculate the density

1. Convert the conductivity to salinity:
2. Calculate the density from S, T, P using UNESCO International Equation of State (1980) UNESCO Technical Papers in Marine Science, Vol. 44

This part of the thesis will present other solutions to measure the water density, one is mapping water density by the dynamics of the robot, and the second one is matching water density with depth by step diving.

5.7.1 Mapping water density by the dynamics of the robot:

Water density from velocity and acceleration:

Dynamics of drifter:

$$Mz'' = (\rho_r - \rho_w)V_r g - \frac{SC}{2}\rho_w (z')^2$$

Thus,

$$\rho_w = \frac{(\rho_r V_r g / M - z'')}{(V_r g + \frac{SC}{2}(z')^2)/M} = \frac{(\rho_r V_r g - Mz'')}{V_r g + \frac{SC}{2}(z')^2}$$

Here M is the total mass of the drifter 42kg for simulation

S is the vertical surface of the drifter, 0.05 m^2

C is the drag coefficient, 1.9

V_r is the volume of the drifter: M / ρ_r

Successive difference method:

$$z_1(t_1) \qquad z'_{12} = v_{12} = \frac{z_2 - z_1}{t_2 - t_1} \qquad z'' = a_2 = \frac{v_{23} - v_{12}}{t_{23} - t_{12}}$$
$$z_2(t_2)$$
$$z_3(t_3) \qquad z'_{23} = v_{23} = \frac{z_3 - z_2}{t_3 - t_2}$$

To compare which method is better to get the water density, result were tested in the simulator: Here the robot was configured to be 1006 kg/m^3 and dives freely in the simulator, the CTD measurements were recorded in log files. Results of both of the two methods are shown in figure 44.

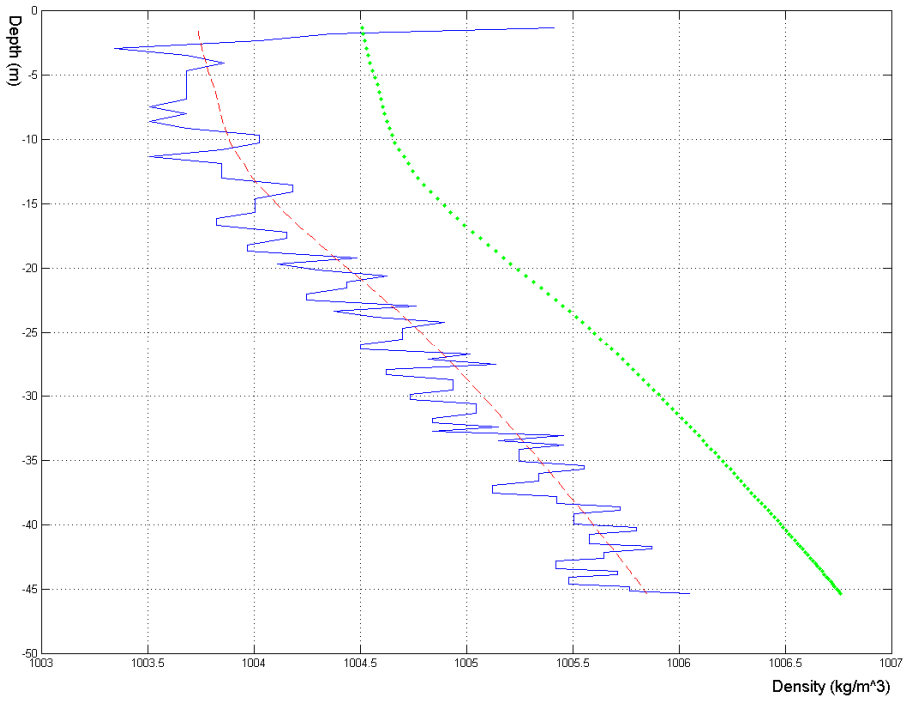

Fig. 44. Density from calculations with +15% conductivity errors

Real line is the water density curve calculated from CTD measurements without conductivity error (theory value). Broken line shows the water density calculated from velocity and acceleration from pressure sensor. Point line is the water density calculated from CTD measurements with +15% conductivity errors. From the figure, the density calculated from dynamics has maximum error about 0.25 kg/m^3, the error is because the velocity and acceleration in the diving process is very small, so that the error to get velocity and acceleration from pressure measurements cannot be neglected. However, the water density calculated from salinity, temperature and

pressure has about 1 kg/ m^3 in the condition of 15% conductivity errors. Therefore, in a certain sense, getting water density from dynamics is a better solution. Assume that the conductivity error is smaller, for example 5%, the result of figure 45 shows that density error is still larger than the result from robot dynamics.

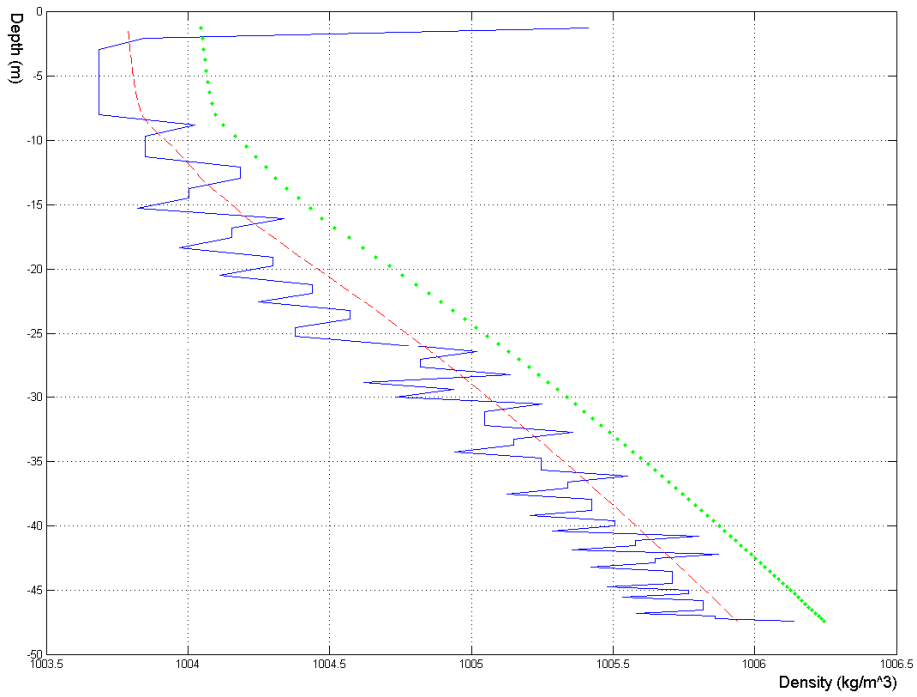

Fig. 45. Density from calculations with +5% conductivity errors

Point line: water density calculated from CTD measurements with +5% conductivity errors. From the result we can conclude that 15% conductivity error is intolerable for the density calculation, and the second figure show the result of 5% conductivity error, which is also intolerable, and if the conductivity sensor error can reach more than 20% as the designer said, it is unusable to get density from CTD measurements by the first method. But the result from the velocity and acceleration has smaller errors. That means if the accuracy of the conductivity sensor cannot be improved, the second method is a tolerable alternative.

To reduce the noise, a mean filter was used. Kalman filter is another natural approach instead of mean filtering (which is off-line). However, the onboard computer of SWARM robot is slow and Kalman filter needs more storage and computation, so that a simpler mean filter is used. As the figure shows, the green points are the

measurements and the red points are the density values after a mean filter length of 9, we can see the after the filter the density approximates the theory value very much, and we can use a polyfitting method to approach the density model, which is showed in Figure 46 and 47.

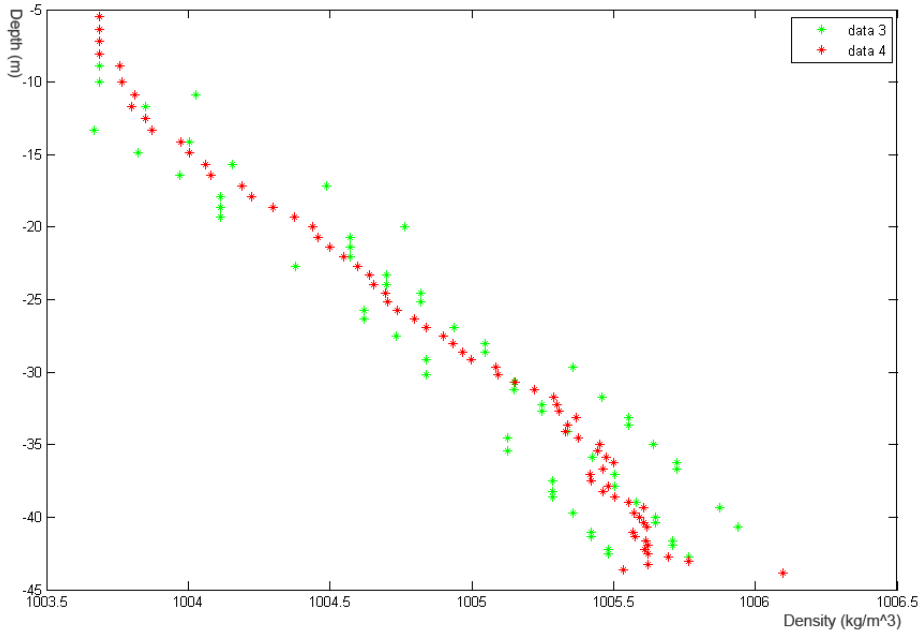

Fig. 46. Density from calculation and after a mean filter

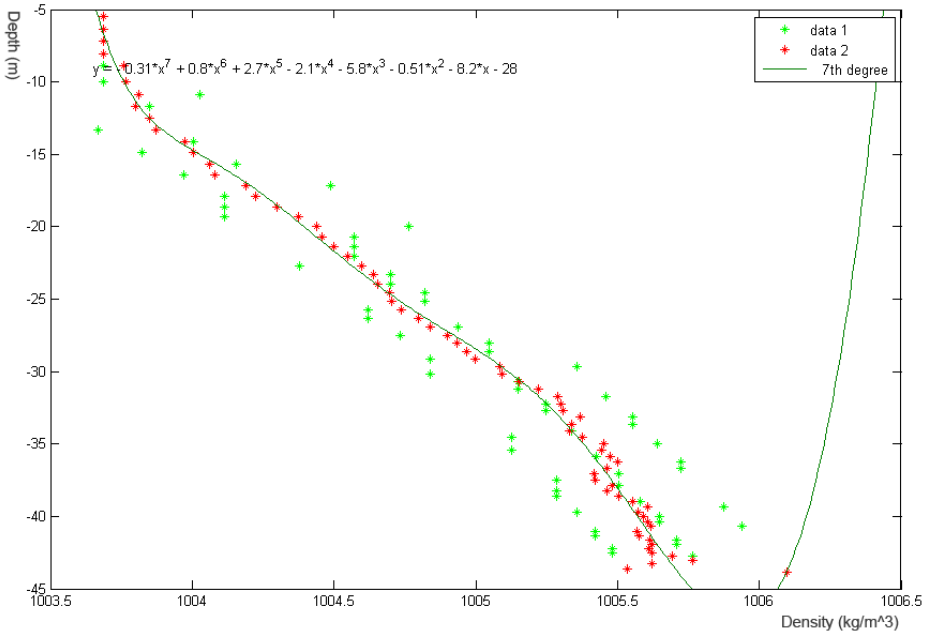

Fig. 47. 7th order polyfitting of the density measurements

5.7.2 Matching density with depth by step diving

As discussed before, the water density can be measured by the robot dynamics, but there might be problems for the real test, because in the calculation the velocity and acceleration of the robot is deduced from the pressure sensor, as acceleration is not easy to get from sensors, a 1.4s pressure sampling interval will cause large acceleration error, the direct density mapping method seems tolerable in the simulator, but it is not reliable in real underwater environments, this should be tested in real test. However, in case the calculation error cannot be trusted in real world, there is another choice: matching water density with depth by step diving

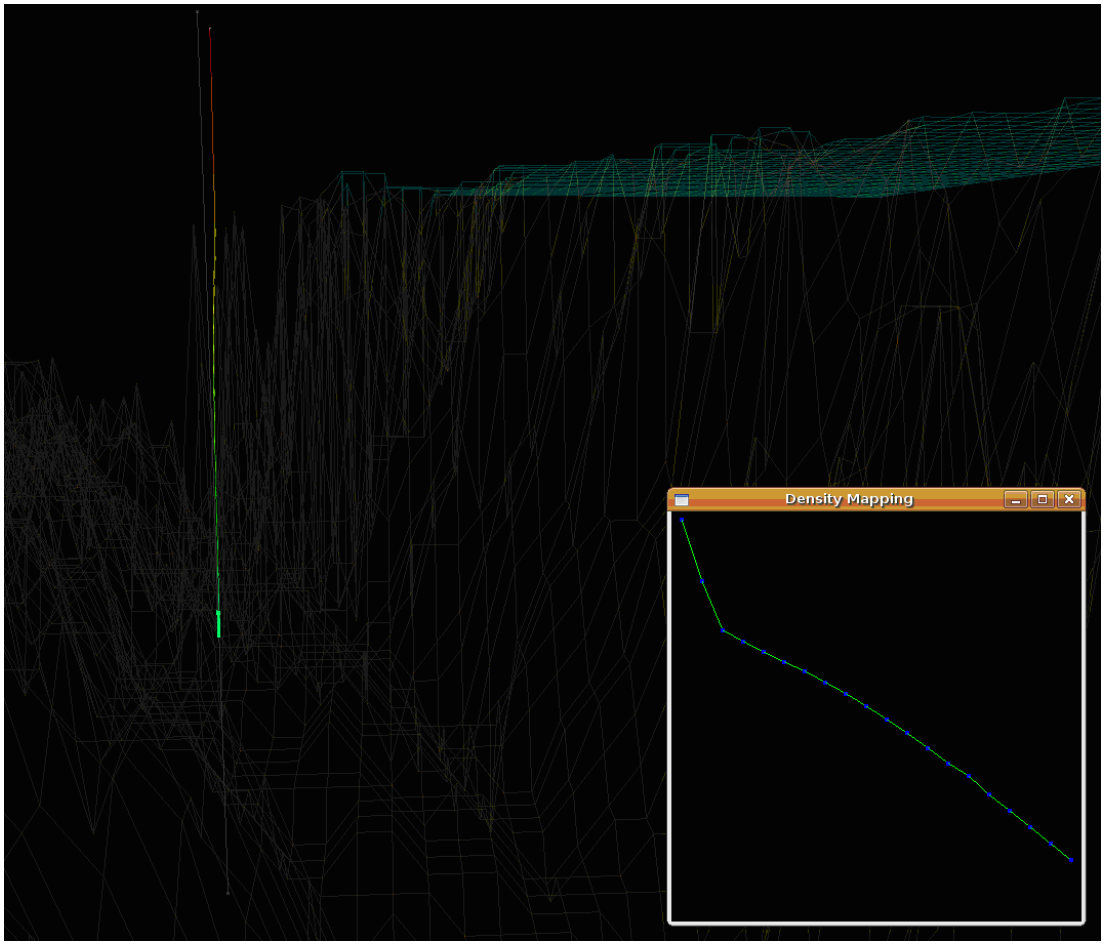

Fig. 48. Matching density with depth by step diving

Figure 48 presents a screenshot of the density matching simulation, the program adds the density of the robot 0.1 kg/cm^3 every step, and when the robot stops, the water density is just equal to the robot density, so that there is no need to use the unreliable acceleration, then the robot add 0.1 kg/cm^3 again and dives another step.

In the simulation of figure 48 showed, there are 20 steps, density add from 1003.5 to 1005.4 every step of 0.1 kg/cm^3 and dives from 1.09 meter to about 40 meters depth. The matching result is showed in the sub window of figure 48, which is plot by OpenGL sub code.

In the simulator it takes about 90 seconds, and the simulator is 200 times fast than the real time. Therefore, to finish this 20 step water density matching, it will take about five hours.

Figure 49 zooms in the robot diving trace.

Fig. 49. Step density-depth matching diving trace

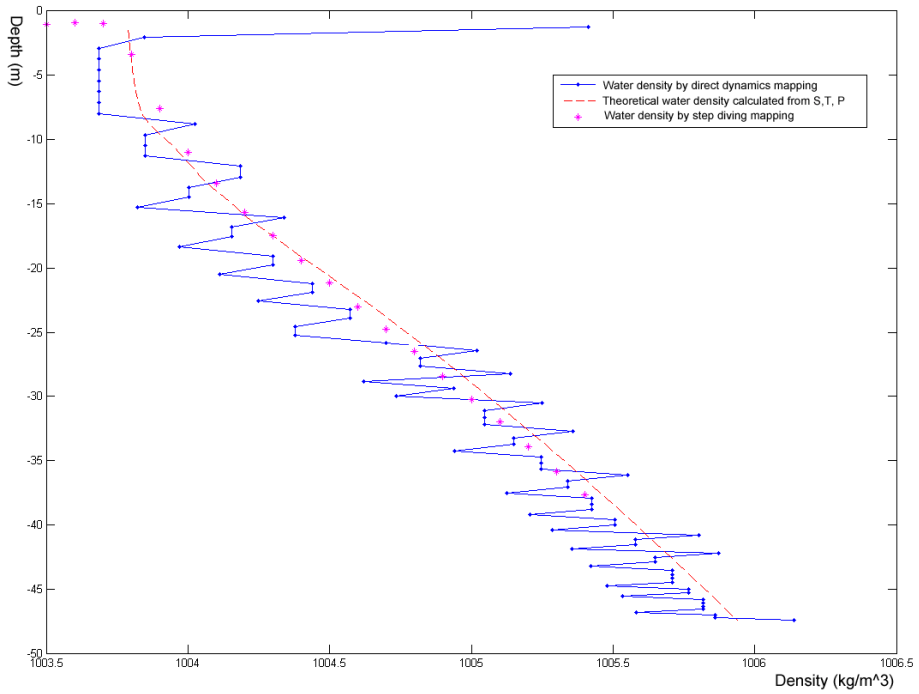

Fig. 50. Water density from direct mapping and density-depth matching

Figure 50 is the result plotted in Matlab compared with the directed mapping mentioned before. The star points are the density from step density-depth matching, as the result showed, step density-depth matching is more reliable and has smaller error. The only drawback of step matching is that it cost more time and energy, because the density of the float add 0.1 kg/cm^3 every step and wait until the float stopped. Because the piston moves for a lot of times, it cost more energy than mapping density directly, however, we need only to measure the water density in the case of we don't have water properties in hand and not necessary to be done before every mission as discussed before that the water density changes not too much in one month.

Chapter 6

Real test plan in Baltic Ocean

It is a pity that the real test of the control algorithm cannot be included in this thesis report, because the real test schedule is later than the author's thesis and oral defense time, so that this chapter will only present the real test plan here:

1. Check the hardware, make sure all the components work properly and check gas tight of the float body shell.
2. Prepare test programs, convert the control code for the simulator to the standard code for the robot system.
3. Calibrate the parameters used in the diving control: total mass of the float, volume of the piston and volume of the entire robot body, drag coefficient and etc.
4. Test hardware and software in the laboratory tank, especially test the piston movement.
4. Compare the real environments with the simulation data. Get density data from direct density mapping and step density mapping, compare the results.
5. Examine and certify the conclusions in the thesis about the environment model, including salinity, temperature and density profile with time and location.
6. Update the seawater density model using the measurement data.
7. Diving test using the new environment based control and evaluate this control method.
8. Test and evaluate the effects of real world disturbances and measurements error and delay, certify how much they affect the control algorithm.
9. Measure the energy consumption in the real diving, estimate the energy budget for the system mission using the now control algorithm.

Chapter 7

Conclusion

"Why do we love the sea? It is because it has some potent power to make us think things we like to think."

--Robert Henri (1865-1929)

This work investigates the state-of-arts of underwater robots, seawater properties and the control of underwater robots. Underwater robots have been drawing increasing amount of interests in various areas such as ocean resource exploitation. The thesis presented an energy optimized environment based diving method for the SWARM underwater project. The general concept of this diving algorithm is using the environment profile for the diving control. The Baltic seawater properties including temperature, salinity, pressure and density are analyzed, the author found out the rules of how those environments changes with time and location, and how can the environment information be used to make estimation in the diving.

Generally, the diving method makes an environment model and tries to estimate the water density at different depths, and then sets the robot density to match the target density, so that the piston need only to move once and can go to a depth very close to the target depth. Every time the piston moves, it needs much energy to activate the motor and conquer the breakout friction that is why we need to reduce the piston movements as much as possible. This diving method can save much more energy than the old control method, and it has less depth error and overshoot. However, it can only control the diving depth, but cannot control the velocity of the robot in the diving process, so that it can be called "environment based depth control of an underwater vehicle".

There are three necessary conditions to use this environment based control. Firstly, water density does not change very much with time, so that in the mission period we can estimate the water density without considering the effect of time, according to the environments study in chapter 4, result shows that water density does not change much in one month, therefore the estimation is practicable for a two weeks mission. Secondly, water density does not change very much with location. As chapter 4 shows, density varies slightly in a large area of Baltic Sea, so that the density estimation does not depend on the location of the float much. Lastly, the overshoot in the diving process is very small because the robot density is quite close to the water density, and the join force on the float is very small. On the other hand, the diving speed is slow enough to take CTD measurements.

In the simulation test, the float can approach target depth within 1 meter error by only one or two piston movements. Result showed very good depth control performances. This depth control method can be used when velocity does not need to be controlled in the diving process, in fact the SWARM robots do not need constant diving velocity. The environment based control can be combined with other dynamics controllers in the underwater missions.

To create a water density model, salinity, temperature and pressure data are required, in the case of low accuracy measurements, water density can also be deduced from diving dynamics. Different methods to get seawater density profile were discussed and evaluated. The first method is to drop the robot into the water and calculate the water density from velocity and acceleration. Velocity and acceleration are calculated from pressure measurements, but the velocity and acceleration are very small and the pressure measurements have a time interval of more than 1.4 seconds, therefore, water density from this method has large error. The other method is to dive step by step, which costs more time and energy to map the water density but has higher reliability and lighter error.

References

[1] Kirby, Geoff. "Navies in Transition: A History of the Torpedo; The Early Days." Originally published in the Journal of the Royal Navy Scientific Service, Vol 27 No 1

[2] Christopher von Alt. "Autonomous Underwater Vehicles" March 24-26, 2003 Prepared for the Autonomous Underwater Lagrangian Platforms and Sensors Workshop, Woods Hole Oceanographic Institution

[3] Ewart, T. E., "Observations from Straight Line Isobaric Runs of SPURV", Joint Oceanography Assembly; Edinburgh (UK), 13 Sep. 1976.

[4] Endicott, D. L., Khul, G. R., "The Fast Area Search System", Naval Command Control and Ocean Surveillance Center, Technical Report 1562, 1992.

[5] Chris J Anastasiou. Presentation of "From mines to manatees: The US navy's autonomous underwater vehicle (AUV) program for rapid environmental monitoring and characterization",

[6] Wikipedia: Argo (oceanography) URL: http://en.wikipedia.org/wiki/Argo_float

[7] Argo homepage: Argo, part of the integrated global observation strategy. URL: http://www.argo.ucsd.edu/FrAbout_Argo.html

[8] Ocean Instruments, RAFOS Float. URL: http://www.whoi.edu/instruments/viewInstrument.do?id=1061#1082

[9] The Seaglider Fabrication Center (SFC) of the University of Washington URL: http://www.seaglider.washington.edu/products.html

[10] Deep Phreatic Thermal Explorer (DEPTHX) project homepage URL: http://www.frc.ri.cmu.edu/depthx/

[11] Jean Kumagai. IEEE Spectrum: "Swimming to Europa. A robot designed to explore Mexican sinkholes is pointing the way to Jupiter's watery moon ", Sep. 2007

[12] Zhao, S.; Yuh, J.， Experimental Study on Advanced Underwater Robot Control，Robotics, IEEE Transactions on，Volume 21, Issue 4, Aug. 2005 Page(s): 695-703

[13] Motion control of intelligent underwater robot based on CMAC-PID, Furong Liu, Information and Automation, 2008. ICIA 2008. International Conference on

[14] Experimental Study on Advanced Underwater Robot Control, Side Zhao, Member, IEEE, and Junku Yuh, Fellow, IEEE, IEEE transactions on robotics, vol. 21, no. 4, August 2005
[15] Wikipedia. "Sliding mode control", URL: http://en.wikipedia.org/wiki/Sliding_mode_control
[16] Dana R. Yoerger and Jean-Jaques E. Slotine. "Robust trajectory control of underwater vehicles". IEEE journal of Ocean Engineering, OE-10(4):462-470,1985
[17] Wikipedia. "Adaptive control", URL: http://en.wikipedia.org/wiki/Adaptive_control
[18] Yang Shi, "Adaptive Depth Control for Autonomous Underwater Vehicles Based on Feedforward Neural Networks", International Journal of Computer Science & Applications, Vol. 4 Issue 3, pp 107-118
[19] Timothy W.McLain, Randal W.Beard, "Nonlinear Optimal Control of an Underwater Robotic Vehicle" 1998, in International Conference on Robotics and Automation
[20] Wikipedia. Fuzzy control system, URL:http://en.wikipedia.org/wiki/Fuzzy_control_system
[21] Min Xu. "Adaptive fuzzy logic depth controller for Variable Buoyancy System of Autonomous Underwater Vehicles" IEEE, Applications to Robotics and Automation, 1994
[22] Webpage of Automation and System department, TKK. URL: http://automation.tkk.fi/swarm
[23] The 210 CTD Logger. www.AquateSubsea.com
[24] SANYO Cadnica DATABOOK. URL: http://batteries.sanyo-component.com/fileadmin/EDITORS/BATTERIES/industrial/datasheets/nicd/KR-5000DEL.pdf
[25] ENSIETA. "SWARM diving system design and test report"
[26] Keil. "Overview of the Keil RTX166 Real-Time Kernel" URL: http://www.keil.com/rtx166/
[27] The Baltic Sea Portal, maintained by the Finnish Environment Institute, the Finnish Meteorological Institute and the Ministry of Environment in Finland.
[28] N.P. Fofonoff and R.C. Millard Jr. "Algorithms for computation of fundamental properties of seawater", Unesco technical papers in marine science vol. 44, 1980

[29] Thomas Collilieux, "Design, Integration and Testing of the float diving system", 08. 2005

Appendix A

A.1 Simulation installation, compilation and running guide

The simulator and diving test code are written in C++ in Linux operation system (e.g. Ubuntu), there are a few libraries need to be installed before compiling and running the simulator, the following instruction is for Ubuntu system.

A1.1 Setup the libraries:

- OpenGL

Install “freeglut” in the Ubuntu system, for example, search “freeglut3” and “freeglut3-dev” in Synaptic Package Manager and install all the other dependency packages

- SVL-1.5

Download SVL from internet, for the source code install, follow the install guild file, and if it happens: “no permission to access usr/local/lib” try to use sudo command like:

sudo mv svl-1.5/lib/libsvl.a usr/local/lib

- GIMI

GIMI or GIM interface, is a C++ interface provide by TKK, used for joining the different modules of the simulator, sending and receiving data from one client to another.

Copy the GIM folder into the swarm directory, notice that the "GIM" folder should be next to "sim" folder, not inside, but make a link of "GIM" inside "sim", make "GIMI/" "util/" "tcpHub/", if there is an error asking for “xml2-configure”, install "libxml2-dev" from the Synaptic Package Manager first.

- Netcdf

NetCDF (network Common Data Form) is an interface for scientific data access and a freely-distributed software library that provides an implementation of the interface. The netCDF library also defines a machine-independent format for representing scientific data. Together, the interface, library, and format support the creation, access, and sharing of scientific data.

In Ubuntu, install “libnetcdf4” and the dependencies from Synaptic Package Manager.

- Curses handling of terminal window

Install libraries for ncurses: “libncurses5-dev”, this package contains the header files, static libraries and symbolic links that ncurses need.

A1.2 Run the simulator

To run the simulator, open 4 terminals in Linux desktop, change the current direction to the swarm/sim directory, if there are no executable applications, use “make all” command to compile the source code. Any change in the .h file, Using commands ‘make clean’ before ‘make all’ is recommended.

Run tcpHub of GIMI:

./GIM/tcpHub/tcpHub –p 50002

Run sssim-env module:

./sssim-env

Run sssim-ctrl module:

./sssim-ctrl

Run the diving test module:

./sssim-drifter-diving or other test program ./sssim-drifer-*

A.2 Introduction of the diving test programs

Here are several diving test programs for different purposes:

sssim-drifter-diving is a program testing the basic diving control algorithm, which reads the input depth from the user, predicts target density and tests diving automatically. Manual diving control is also working within this program.

sssim-drifter-mapping is used for mapping the water density, and there are two options to make density profile: directly mapping and step mapping.

sssim-drifter-overshoot is a program used for test the effects of drag coefficient, the maximum diving velocity, overshoot, and settling time can be get from the diving tests. Here the settling time is determined by the user pressing ‘q’ when the robot is settling down, which is because that it is difficult to get the diving amplitude from the CTD measurements.

All the programs save diving data to log files, data processing can use Matlab.

A.3 Keyboard functions in the simulator:

- Switch to the Baltic window, which has the 3D graph plot of the ocean

Press space bar to start the simulation time, press again to pause the simulation time.

Press ‘V’ key to change the current color view between salinity, temperature and pressure.

Press ‘B’ key to change the background of the sea bottom, there are three different views: gridding, solid drawing and no background.

Press Tab key to adjust the view to the robot, if there are more than one robots in the simulator, tab key will switch view between the robots.

Press ‘F’ key to plot the three dimensional sea current flows by suspending points, press ‘0’ key to regress all the floating points.

In the diving test window:

Press up and down arrow to manually control the robot; this will change the density of the robot by a small value every time, the middle rolling key of the mouse has the same function.

- In the sssim-drifter-diving program

Press ‘D’ to start automatic diving and input the target depth. The robot will estimate the target density and adjust its density to get to the target depth. All diving information including time, depth, CTD measurements and altimeter measurements will save to a log file in the swarm folder.

Press ‘M’ key to start a density mapping mission: the robot goes to the surface and change its density to a certain value and dives freely. The velocity, acceleration will be deduced from pressure measurements and the water density will be calculated and save to the log file. In the step mapping, it does the same calculation but the density is changed step by step, the water density is calculated in every step when the robot stops.

A.4 Source code configuration

- In the automatic diving test, simulator is set to pick up a random location and a random time but in the drag coefficient test the location and time is fixed.

Random location is set in ‘drifer.cpp’:

```
void Drifter::RandomizeWithCenter() {
double
        sea_w = (sea->nx-1) * sea->var_pos[0][1],
        sea_h = (sea->ny-1) * sea->var_pos[1][1];
do {
        pos[0] = sea->var_pos[0][0] + sea_w * ( 0.45 + 0.10 * rand() / RAND_MAX );
        pos[1] = sea->var_pos[1][0] + sea_h * ( 0.30 + 0.10 * rand() / RAND_MAX );
```

```
            pos[2] = FLOAT_HALF_HEIGHT;
} while ( !UpdateEnvironment(true) );
            volume = FLOAT_MASS / rho;
            time = sea->t_now;
            //pos[2] = FLOAT_HALF_HEIGHT;
            vel = vl_zero;
            UpdateEnvironment(true);
}
```

In the drag coefficient test, this should be fixed location, it should change to:

```
            pos[0] = sea->var_pos[0][0] + sea_w * ( 0.5 );
            pos[1] = sea->var_pos[1][0] + sea_h * ( 0.3 );
```

Random start time is set in ‘sssim-env.cpp’:

```
time_now = 24 * 3600 * 25.0 * rand() / RAND_MAX;
```

In the drag coefficient test, change this to

```
time_now = 0;
```

In the drag coefficient test, the drag coefficient is set in ‘drifter.h’

```
#define  FLOAT_VERTICAL_DRAG_COEF  1.9
```

- Simulator is speedup by a time factor; the default value is 200, it means the simulation time is 200 times faster than the real time. Change the simulation time factor in ‘sssim-env’:

```
#define  REALITY_MULTIPLIER  200.0
```

Appendix B

Matlab program guide

All the Matlab script need to load the salinity and temperature data to the workspace before running, make sure that S.mat and T.mat is in the current work directory, so that the programs can run directly or you can use load command to load them into workspace.

BalticCTD.m: a program can plot salinity, temperature and density curves with approximate mathematical models. Parameters can be changed from the user interface.
Sali_time_step.m: plots the salinity data with time step. It displays the salinity with time by animation, a Sali_time_step.gif animation file will be saved when the program is running.
Temp_time_step.m: same function as above, but display temperature data.
Deinsity_time_step.m: calculate density and display density curves with time.
Baltic_location_step.m: a program displays the water properties with longitude and latitude
All_time_step.m: plot water density, salinity and temperature together with time.
All the programs are used to display the 4 dimensional data. The Matlab programs can be change to plot curve of seawater properties of any time and any location, they are used to find out the rules of how the environments changing, and try to model the water properties. Not all the programs have user interface, to change the parameters, x refers to longitude (ordinal number, from 1-17), y refers to latitude (ordinal number, from 1-17), z refers to depth (ordinal number, from 1-21), and t refers to time (ordinal number, from 1-124).

Appendix C

The simulator and diving test code have about ten source files, the “sssim-drifter-diving.cpp” here attached is the most important code file for the diving algorithm test, which consists of the diving control algorithm mentioned in this thesis.

```
// File name: sssim-drifter-diving.cpp

#include <cstdlib>
#include <cstdio>
#include <csignal>
#include <ctime>
#include <curses.h>
#include "sssim.h"
#include "drifter.h"
#include <GL/glut.h>
#include "util.h"

Sea * baltic;

double density = 0.0;
	double depth[20];

volatile bool
	active = true,
	runtime = false;

gimi::GIMI * gimipointer = NULL;

unsigned long time_now = 0;
DrifterCTD ctd;
unsigned char altim;

//--------------------------------------------------------------------------
static void sig_handler( int signumber, siginfo_t *info, void *context ) {
	endwin();
	putchar('\n');
	switch( info->si_signo ) {
		case SIGINT:	printf( "Got SIGINT > exiting.\n" );			break;
		case SIGTERM:	printf( "Got SIGTERM > exiting.\n" );
	break;
		default:		printf( "Got signal %d\n", info->si_signo );	return;
	}
	active = false;
	if ( gimipointer != NULL ) gimipointer->stop();
	else printf( "Could not find GIMI-object to stop.plot library Receive/ping/serviceDiscovery-
methods won't quit until timeout.\n" );
}

//--------------------------------------------------------------------------
int tell_env( int type, const char * data = NULL, int len = 0 ) {
	static int msg_counter = 0;
	if ( data && !len ) len = strlen(data);
	int gimi_result = gimipointer->send( SSS_ENV_NAME, data, len, SSS_GIMI_MSG_ID, type,
msg_counter++ );
	if ( gimi_result != GIMI_OK ) printw( "TX (%s) ERROR: %s\n",
sss_msg_type_to_str(type).c_str(), gimi::getGimiErrorString(gimi_result).c_str() );
	refresh();
```

```
	return gimi_result;
}

//---------------------------------------------------------------------------
bool sss_get_altimeter( unsigned char &altim ) {
	gimi::GIMIMessage msg;
	int gimi_result;

	gimi_result = tell_env( SSS_DATA_ALTIMETER );
	if ( gimi_result != GIMI_OK ) return FALSE;

	gimi_result = gimipointer->receive( msg, -1, SSS_GIMI_MSG_ID, SSS_DATA_ALTIMETER,
0 );
	if ( ( gimi_result != GIMI_OK ) || msg.datalength != sizeof(unsigned char)  ) return FALSE;

	altim = *( reinterpret_cast< unsigned char * >( msg.data ) );
//	printw( "%lu: %02X\n", altim );refresh();
	return TRUE;
}

//---------------------------------------------------------------------------
bool sss_get_ctd( DrifterCTD &ctd ) {
	gimi::GIMIMessage msg;
	int gimi_result;

	gimi_result = tell_env( SSS_DATA_CTD );
	if ( gimi_result != GIMI_OK ) return FALSE;

	gimi_result = gimipointer->receive( msg, -1, SSS_GIMI_MSG_ID, SSS_DATA_CTD, 0 );
	if ( ( gimi_result != GIMI_OK ) || msg.datalength != sizeof(DrifterCTD)  ) return FALSE;

	ctd = *( reinterpret_cast< DrifterCTD* >( msg.data ) );
//	printw( "CTD %lu: %f C, %u mbar, %f mS/cm\n", ctd.time, ctd.temperature, ctd.pressure,
ctd.conductivity );refresh();

	return TRUE;
}

//------------------STD prediction from a target depth-------------------------
double salinity_prediction(double depth){
	double z2 = 65.0;

	return -0.03*(depth-z2)+7.0;
}

double temperature_prediction(double depth){
	double z1 = 0.0,z2 = 65.0;
	double t1 = 16.0,t2 = 4.0;
	double k = 18.0;

	z1=z2-sqrt(t1-t2)*k;

	if(0<depth&&depth<z1)	return t1;
	else if(z1<=depth&&depth<=z2)	return ((z2-depth)/k)*((z2-depth)/k)+t2;
	else if(depth>z2) return -0.01*(depth+z2)+t2;
	else return 0.0;
}

long unsigned pressure_prediction(double depth){

	return pressure_from_depth(depth)+1013.25;
}

double density_prediction(double depth){
	static double
		top[] = { -0.01458064, 1003.65416396 },
		bot[] = { 5.82791451e-13, 2.93618412e-10, 5.80359192e-08, 5.74515221e-06,
```

```
3.01506402e-04, 7.94346722e-03, 3.38527732e-02, 1003.71941686 };
	double
		density_pr = 0.0,
		* pa;
	unsigned int i, pa_size;

	if ( depth < 6 ) {
		pa = &top[0];
		pa_size = sizeof(top) / sizeof(double);
	} else {
		pa = &bot[0];
		pa_size = sizeof(bot) / sizeof(double);
	}

	for ( i = 0; i < pa_size; ++i ) density_pr = pa[i] - depth * density_pr;

	return density_pr;
}

//---------------------------------------------------------------------------
void sss_diving( void ){
	double d, depth_now, adj, err, adj_, err_=0;
	int i=1;

	FILE *fp;
	fp= fopen("diving_log","wd");//creat a log file to record the diving data
	if ( fp== NULL ) { exit(-1);}
	fprintf(fp,"time temperature pressure conductivity\n");

	printw("\ninput target depth:\n");refresh();
	std::cin>>d;
	double temperature_pr = temperature_prediction(d);
	double salinity_pr    = salinity_prediction(d);
	long unsigned pressure_pr    = pressure_prediction(d);
	double density_pr     = density_prediction (d);

	density_pr = density_prediction(d)+0.1;printw("\ndensity prediction add 0.1kg/m^3 error(for
testing)\n");//!!!!!add estimation error to test adjustment, fix here

	double k1;


	printw("%dm\n\nprediction:\n\ndensity: %f\n", (int)d,density_pr);refresh();
	printw("%f C, %u mbar, %f mS/cm\n", temperature_pr,pressure_pr,salinity_pr);refresh();
	printw("\nhere we go!\n", (int)d);refresh();

	tell_env( SSS_DATA_DENSITY, reinterpret_cast< char* >( &density_pr ), sizeof(density) );
	DrifterCTD ctd[3];
	bool abort=FALSE, arrive=FALSE, stop=FALSE;
	while(!arrive&&!abort){

		while(!stop){
			sss_get_altimeter(altim);
			//get altimeter and avoid collision, if altimeter is less than 8 meters, float up,
set abort true
			if(altim<80){
				abort=TRUE;
				printw( "\nMy God! We are going to crash! Let's go back to Mommy!
\n");refresh();
				density_pr=1000;
				tell_env( SSS_DATA_DENSITY, reinterpret_cast< char*
>( &density_pr ), sizeof(density) );break;
			}
			//read ctd sensor for three times
			for(int i=0;i<3;i++){
				sss_get_ctd(ctd[i]);
				printw(".");refresh();
```

```
                                        fprintf(fp,"%u %f %u %f\n", ctd[i].time, ctd[i].temperature,
ctd[i].pressure, ctd[i].conductivity);
                                        sleep(0.5);
                                }
                                //check if the drifter stoped
                                if((ctd[1].pressure-ctd[0].pressure)*(ctd[2].pressure-ctd[1].pressure)<=0){
                                        printw( "\n\ndiving stopped, checking depth..\n\n");refresh();
                                        stop=TRUE;
                                }
                        }

                        if((ctd[2].pressure<pressure_pr-50||ctd[2].pressure>pressure_pr+50)&&!abort) {
                                printw( "not close to target depth, adjusting...\n");refresh();
                                depth_now=depth_from_pressure(ctd[2].pressure-
STANDARD_ATMOSPHERIC_PRESSURE);
                                err = depth_now-depth_from_pressure(pressure_pr-
STANDARD_ATMOSPHERIC_PRESSURE);
                                printw( "\n%f %f \n",depth_now, d);refresh();

                                if(depth_now>0&&depth_now<=18) k1=0.0167;
                                if(depth_now>18&&depth_now<=38) k1=0.0600;
                                if(depth_now>38&&depth_now<=82) k1=0.0295;
                                if(depth_now>82&&depth_now<=160) k1=0.0189;
                                if(depth_now<d) k1=-k1;

                                if(i==1) {adj=-k1*err; i++;}
                                else
                                adj = -adj_ * err / (err - err_);
                                printw("%f", -adj_/(err - err_));
                                adj_=adj, err_=err;
                                density_pr+=adj;
                                tell_env( SSS_DATA_DENSITY, reinterpret_cast< char* >( &density_pr ),
sizeof(density) );
                                sleep(1);

                                printw( "\ndensity changed %f kg/m^3\n",adj);refresh();
                                stop=FALSE;
                        }
                        else if(!abort){
                                printw("\nAha! Here we are!\n",int(d));
                                sss_get_ctd(ctd[0]);
                                fprintf(fp,"%u %f %u %f\n", ctd[0].time, ctd[0].temperature, ctd[0].pressure,
ctd[0].conductivity );
                                printw( "\n%f C, %u mbar, %f mS/cm\n", ctd[0].temperature, ctd[0].pressure,
ctd[0].conductivity );refresh();
                                depth_now=depth_from_pressure(ctd[0].pressure-
STANDARD_ATMOSPHERIC_PRESSURE);
                                printw( "\nfinal depth:%fm\ntarget depth:%fm\ndiving
error:%fm\n",depth_now, double(d), depth_now-d);refresh();
                                arrive= TRUE;
                        }
                }
                printw( "\nCheck 'diving_log' for more information!\n");refresh();
                fclose(fp);//close log file

//              return TRUE;
}


//---------------------------------------------------------------------------
void draw_density(void)
{
                glClear(GL_COLOR_BUFFER_BIT);
                glColor4f(0.0,0.5,0.5,0.5);
                glMapGrid2f(20,0.0,1.0,20, 0.0,1.0);

                for(int i=0;i<20;i++){
```

```
                glPointSize(4.0);
                glBegin(GL_POINTS);
                glColor4f(0.0,0.0,1.0,1.0);
                glVertex2f(-0.95+i*0.1,1-depth[i]*0.04);
                glEnd();

                glLineWidth (0.1);
                glBegin(GL_LINES);
                glColor4f(0.0,1.0,0.0,1.0);
                if(i!=19){
                glVertex2f(-0.95+(i)*0.1,1-depth[i]*0.04);
                glVertex2f(-0.95+(i+1)*0.1,1-depth[i+1]*0.04);}
                glEnd();
        }
        glFlush();
}

void keyboard(unsigned char key, int x, int y){
switch(key){
        case  27:
        case 'q': exit(0); break;
        }
}

/*
void SpecialKeys(int key, int x, int y){
if(key == 27)
        exit(0);
}
*/
//------------------------------------------------------------------------
void sss_mapping(int argc, char **argv){

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
        glutInitWindowPosition(100, 100);
        glutInitWindowSize(400, 400);
        glutCreateWindow("Density Mapping");


        printw("Mapping water density...\n");refresh();

        FILE *fp;
        if((fp=fopen("mapping_log", "wb"))==NULL) {
                printf("Cannot open file.\n");
        exit(1);
        }
        for(int i=0;i<20;i++){
                sss_get_altimeter(altim);
                if(altim<40){
                        printw( "\nWe are going to crash! Mission
aborted!\n");refresh();density=1000;
                        tell_env( SSS_DATA_DENSITY, reinterpret_cast< char* >( &density ),
sizeof(density) );break;
                        }
                density=1003.50+0.1*i;
                tell_env( SSS_DATA_DENSITY, reinterpret_cast< char* >( &density ),
sizeof(density) );
                sleep(3);
                sss_get_ctd(ctd);
        fprintf(fp, "depth %f  density %f\n", depth_from_pressure( ctd.pressure-
STANDARD_ATMOSPHERIC_PRESSURE), density );
        printw("depth %f  density %f\n", depth_from_pressure( ctd.pressure-
STANDARD_ATMOSPHERIC_PRESSURE), density );
        depth[i]=depth_from_pressure( ctd.pressure-STANDARD_ATMOSPHERIC_PRESSURE);
        }
```

```
	fclose(fp);// end log file

	glutDisplayFunc(&draw_density);
	glutKeyboardFunc(keyboard);
//	glutSpecialFunc(SpecialKeys);
	glutIdleFunc(sss_diving);
	glutMainLoop();//draw density profile
//	return TRUE;
}

//--------------------------------------------------------------------------
int main( int argc, char **argv ) {
	// signal handling
	struct sigaction siga;
	sigemptyset( &siga.sa_mask );
	siga.sa_flags = SA_SIGINFO;
	siga.sa_sigaction = sig_handler;
	sigaction( SIGINT, &siga, NULL );
	sigaction( SIGTERM, &siga, NULL );

	gimi::GIMI gi(GIMI_DEFAULTQUEUESIZE);
	gi.addAcceptedService( SSS_GIMI_MSG_ID );

	gimi::GIMIMessage gimi_rx_msg;
	int gimi_result;

	std::cout << "Connecting to GIMI tcpHub " SSS_HUB_NAME ":" << SSS_HUB_PORT << "... "
<< std::flush;
	if ( !gi.connectToHub( SSS_HUB_NAME, SSS_HUB_PORT, SSS_CTRL_NAME ) ) {
		std::cout << "ERROR!\n";
		exit(-1);
	}
	std::cout << "ok, got id " << gi.getOwnId() << '\n';

	std::cout << "Pinging environment " SSS_ENV_NAME "..." << std::flush;
	while (true) {
		gimi_result = gi.ping( SSS_ENV_NAME, 500 );
		if ( gimi_result >= 0 ) {
			std::cout << " rtt " << gimi_result << "ms, id " << std::flush;
			int env_id = gi.findClientId( SSS_ENV_NAME );
			if ( env_id <= 0 ) {
				std::cout << "ERROR: " << gimi::getGimiErrorString(gimi_result)
<< '\n';
				gi.stop();
				exit(-1);
			}
			std::cout << env_id << '\n';
			break;
		} else if ( gimi_result == GIMI_TIMEOUT ) {
			std::cout << '.' << std::flush;
		} else {
			std::cout << " ERROR: " << gimi::getGimiErrorString(gimi_result) << '\n';
			gi.stop();
			exit(-1);
		}
	}
	gimipointer = &gi;

	std::cout << "Announcing presence ... " << std::flush;
	gimi_result = gimipointer->send( SSS_ENV_NAME, NULL, 0, SSS_GIMI_MSG_ID,
SSS_CTRL_NEW_DRIFTER, 0 );
	if ( gimi_result == GIMI_OK ) std::cout << "ok.\n";
	else {
		std::cout << "ERROR: " << gimi::getGimiErrorString(gimi_result) << '\n';
		exit(-1);
	}
```

```
	initscr();
	cbreak();
	noecho();
	keypad( stdscr, TRUE );
	nodelay( stdscr, TRUE );
	idlok( stdscr, TRUE );
	scrollok( stdscr, TRUE );

	bool
		req_altim = true,
		req_ctd = true;
	clock_t
		req_altim_ct = 0,
		req_ctd_ct = 0;

	while( active ) {
		gimi_result = gi.receive( gimi_rx_msg, 0 );
		bool msg_ok = true;
		switch (gimi_result) {
			case GIMI_OK:
				switch( gimi_rx_msg.minorTypeId ) {
					case SSS_CTRL_EXIT:
						raise(SIGTERM);
						break;
					case SSS_CTRL_RESET:
						printw( "==== RX reset\n" );
						break;
					case SSS_CTRL_START:
						printw( "==== Time started\n" );
						runtime = true;
						break;
					case SSS_CTRL_PAUSE:
						printw( "==== Time paused\n" );
						runtime = false;
						break;
					case SSS_DATA_CTD:
						msg_ok = req_ctd = sss_get_ctd( ctd );
						break;
					case SSS_DATA_ALTIMETER:
						msg_ok = req_altim = sss_get_altimeter( altim );
						break;
					case SSS_DATA_DENSITY:
						density = *( reinterpret_cast< double *
>( gimi_rx_msg.data ) );
						printw( "\n-------Simulation of drifter diving-----
\n" );
						printw( "current water density is: %f\n", density );
						printw( "press m to dive and make density
mapping\n" );
						printw( "press up or down arrow for manual
diving\n" );
						printw( "press d key to start an automatic
diving\n" );
						printw( "----------------------------------------\n\n" );
						break;
					case SSS_DATA_MESSAGE:
						printw( "%lu: RX msg (%d) ",
gimi_rx_msg.messageId, gimi_rx_msg.datalength );
						for ( int i = 0; i < gimi_rx_msg.datalength; ++i )
addch( gimi_rx_msg.data[i] );
						addch('\n');
						break;
					default:
						msg_ok = false;
				}
				if (!msg_ok) printw( " << %s #%d from %d len %d\n",
					sss_msg_type_to_str( gimi_rx_msg.minorTypeId ).c_str(),
```

```
gimi_rx_msg.messageId, gimi_rx_msg.senderId, gimi_rx_msg.datalength );
                                        break;
                                case GIMI_NOMSGS:                       break;
                                case GIMI_INTERRUPT:   printw( "RX interrupted!" ); break;
                                default:                printw( "ERROR %d (%s) from RX", gimi_result,
gimi::getGimiErrorString(gimi_result).c_str() );
                        }
                        refresh();

                        if ( runtime && req_ctd && ( clock() >= req_ctd_ct ) ) {
                                tell_env( SSS_DATA_CTD );
                                req_ctd_ct = clock() + 0.1 * CLOCKS_PER_SEC;
                                req_ctd = false;
                        }

                        if ( runtime && req_altim && ( clock() >= req_altim_ct ) ) {
                                tell_env( SSS_DATA_ALTIMETER );
                                req_altim_ct = clock() + 0.2 * CLOCKS_PER_SEC;
                                req_altim = false;
                        }

                        int ch = getch();
                        switch( ch ) {
                                case 'q':
                                        endwin();
                                        gimipointer->stop();
                                        exit(0);
                                        break;
                                case '1':
                                        tell_env( SSS_DATA_MESSAGE );
                                        break;
                                case '2':
                                        tell_env( SSS_DATA_MESSAGE, std::string("ping!").c_str() );
                                        break;
                                case KEY_DOWN:
                                case KEY_UP:
                                        density += (ch==KEY_DOWN) ? 0.05 : -0.05;
                                        printw("D %f\n",density); refresh();
                                        tell_env( SSS_DATA_DENSITY, reinterpret_cast< char*
>( &density ), sizeof(density) );
                                        break;
                                case 'm':
                                        sss_mapping( argc, argv );
                                        break;
                                case 'd':
                                        sss_diving();
                                        break;
                                case ERR:
                                        break;
                                default:
                                        printw("\n%c\n",ch);
                                        refresh();
                                        break;

                        }
                }

                endwin();
                return 0;
}
```

Appendix D

D.1 Example diving log files

One of the diving test logs and CTD measurements in simulation:

```
time,depth,temperature,pressure,conductivity

670783 1.079169 15.451266 1121 9.006406
670785 1.200197 15.451262 1133 9.006418
670787 1.351481 15.451259 1148 9.006434
670789 1.522937 15.451255 1165 9.006452
670791 1.714564 15.451225 1184 9.006466
670793 1.906191 15.450707 1203 9.006369
670795 2.107904 15.450162 1223 9.006268
670797 2.319701 15.449602 1244 9.006164
670799 2.531499 15.449031 1265 9.006058
670801 2.743296 15.448454 1286 9.005950
670803 2.955093 15.447873 1307 9.005842
670805 3.166889 15.447290 1328 9.005733
670807 3.378686 15.446706 1349 9.005624
670809 3.590482 15.446121 1370 9.005515
670811 3.802278 15.445536 1391 9.005405
670813 4.024159 15.444951 1413 9.005297
670815 4.235955 15.444366 1434 9.005188
670817 4.447750 15.443495 1455 9.005017
670819 4.659545 15.442562 1476 9.004833
670821 4.871340 15.441630 1497 9.004649
670823 5.083134 15.440698 1518 9.004465
670825 5.294929 15.439767 1539 9.004281
670827 5.506723 15.438836 1560 9.004098
670829 5.718517 15.437906 1581 9.003914
670831 5.940396 15.436976 1603 9.003732
670833 6.152189 15.436047 1624 9.003549
670835 6.363982 15.434776 1645 9.003294
670837 6.575775 15.432504 1666 9.002829
670839 6.787568 15.430233 1687 9.002364
670841 6.999360 15.427963 1708 9.001899
670843 7.211152 15.425695 1729 9.001435
670845 7.422944 15.423428 1750 9.000971
670847 7.634736 15.421163 1771 9.000507
670849 7.846527 15.418899 1792 9.000044
670851 8.058319 15.416636 1813 8.999581
670853 8.270110 15.414375 1834 8.999118
670855 8.481900 15.400994 1855 8.996390
670857 8.683606 15.381478 1875 8.992410
670859 8.895396 15.361986 1896 8.988438
670861 9.107186 15.342523 1917 8.984471
670863 9.318976 15.323089 1938 8.980512
670865 9.530766 15.303686 1959 8.976559
670867 9.732470 15.284314 1979 8.972612
670869 9.944259 15.264974 2000 8.968673
670871 10.156048 15.245665 2021 8.964741
670873 10.357751 15.226389 2041 8.960815
670875 10.569540 15.098736 2062 8.936278
670877 10.771243 14.965327 2082 8.910658
670879 10.972946 14.832964 2102 8.885251
670881 11.174648 14.701752 2122 8.860077
670883 11.376351 14.571755 2142 8.835148
670885 11.578053 14.443008 2162 8.810470
670887 11.769670 14.315530 2181 8.786047
670889 11.961287 14.189331 2200 8.761880
670891 12.152903 14.064409 2219 8.737969
670893 12.344520 13.940763 2238 8.714314
670895 12.536136 13.818387 2257 8.690913
670897 12.717667 13.697275 2275 8.667763
670899 12.909282 13.577418 2294 8.644865
670901 13.090813 13.462867 2312 8.623572
670903 13.262258 13.364016 2329 8.607350
670905 13.443788 13.266157 2347 8.591287
670907 13.625319 13.169275 2365 8.575379
670909 13.796763 13.073355 2382 8.559623
670911 13.968208 12.978390 2399 8.544019
670913 14.139653 12.884368 2416 8.528566
670915 14.311097 12.791281 2433 8.513262
670917 14.482541 12.699124 2450 8.498106
670919 14.643901 12.607889 2466 8.483098
670921 14.805260 12.517571 2482 8.468235
670923 14.976703 12.428162 2499 8.453520
670925 15.138062 12.339657 2515 8.438949
670927 15.289336 12.252052 2530 8.424521
670929 15.450694 12.165341 2546 8.410238
670931 15.612053 12.079517 2562 8.396098
670933 15.763326 11.994575 2577 8.382098
670935 15.914599 11.910511 2592 8.368240
670937 16.065872 11.827318 2607 8.354523
670939 16.217145 11.744993 2622 8.340946
670941 16.368418 11.663528 2637 8.327508
670943 16.509606 11.591744 2651 8.316988
670945 16.660879 11.521174 2666 8.306718
670947 16.802066 11.451336 2680 8.296548
670949 16.943254 11.382223 2694 8.286476
670951 17.084441 11.313829 2708 8.276502
670953 17.225629 11.246148 2722 8.266627
670955 17.366816 11.179176 2736 8.256848
670957 17.497918 11.112908 2749 8.247165
670959 17.629020 11.047341 2762 8.237579
670961 17.770207 10.982470 2776 8.228090
670963 17.901309 10.918292 2789 8.218696
670965 18.032411 10.854803 2802 8.209397
670967 18.163513 10.792001 2815 8.200193
670969 18.284530 10.729882 2827 8.191083
670971 18.415632 10.668442 2840 8.182069
670973 18.536649 10.607679 2852 8.173148
670975 18.657666 10.547589 2864 8.164321
670977 18.788767 10.488169 2877 8.155589
670979 18.909784 10.429415 2889 8.146950
670981 19.020716 10.371327 2900 8.138403
670983 19.141732 10.313901 2912 8.129949
670985 19.262749 10.257132 2924 8.121589
670987 19.373680 10.201018 2935 8.113320
670989 19.484612 10.145558 2946 8.105143
670991 19.605628 10.090747 2958 8.097059
670993 19.716560 10.038999 2969 8.089609
670995 19.827491 10.000886 2980 8.085172
670997 19.928338 9.963201 2990 8.080780
670999 20.039269 9.925933 3001 8.076434
671001 20.150200 9.889070 3012 8.072131
671003 20.251047 9.852605 3022 8.067870
671005 20.351893 9.816533 3032 8.063650
671007 20.462824 9.780847 3043 8.059474
671009 20.563670 9.745543 3053 8.055338
671011 20.664517 9.710619 3063 8.051242
671013 20.765363 9.676071 3073 8.047188
671015 20.866209 9.641896 3083 8.043174
671017 20.956971 9.608094 3092 8.039199
671019 21.057817 9.574661 3102 8.035266
671021 21.148578 9.541596 3111 8.031372
671023 21.249424 9.508898 3121 8.027519
671025 21.340185 9.476564 3130 8.023705
671027 21.430947 9.444595 3139 8.019932
671029 21.521708 9.412988 3148 8.016198
671031 21.612469 9.381743 3157 8.012505
671033 21.703230 9.350859 3166 8.008851
671035 21.793991 9.320334 3175 8.005237
671037 21.874668 9.290169 3183 8.001663
671039 21.965429 9.260361 3192 7.998129
671041 22.046105 9.230910 3200 7.994634
671043 22.136866 9.201815 3209 7.991180
671045 22.217543 9.173076 3217 7.987766
671047 22.298219 9.144691 3225 7.984391
671049 22.378895 9.116660 3233 7.981056
671051 22.459571 9.088982 3241 7.977760
671053 22.540248 9.061656 3249 7.974505
671055 22.610839 9.034682 3256 7.971289
671057 22.691516 9.008059 3264 7.968114
671059 22.772192 8.981785 3272 7.964978
671061 22.842783 8.955861 3279 7.961881
671063 22.913375 8.930286 3286 7.958825
671065 22.994051 8.905058 3294 7.955809
671067 23.064642 8.880178 3301 7.952832
671069 23.135234 8.855644 3308 7.949895
671071 23.205825 8.831457 3315 7.946998
671073 23.266332 8.807615 3321 7.944139
```

671075 23.336924 8.784118 3328 7.941322
671077 23.407515 8.760964 3335 7.938544
671079 23.468022 8.738154 3341 7.935805
671081 23.538613 8.715687 3348 7.933107
671083 23.599120 8.693561 3354 7.930448
671085 23.659627 8.671778 3360 7.927828
671087 23.720134 8.650335 3366 7.925249
671089 23.780640 8.629232 3372 7.922709
671091 23.841147 8.608469 3378 7.920209
671093 23.901654 8.588045 3384 7.917748
671095 23.962161 8.567960 3390 7.915328
671097 24.012583 8.548213 3395 7.912946
671099 24.073089 8.528803 3401 7.910605
671101 24.123512 8.509730 3406 7.908302
671103 24.184018 8.490994 3412 7.906041
671105 24.234440 8.472594 3417 7.903818
671107 24.284863 8.454529 3422 7.901634
671109 24.335285 8.436799 3427 7.899491
671111 24.385707 8.421300 3432 7.897840
671113 24.436129 8.409670 3437 7.897072
671115 24.486551 8.398262 3442 7.896318
671117 24.526889 8.387073 3446 7.895578
671119 24.577311 8.376102 3451 7.894852
671121 24.617649 8.365345 3455 7.894138
671123 24.668071 8.354801 3460 7.893440
671125 24.708408 8.344468 3464 7.892753
671127 24.748746 8.334346 3468 7.892080
671129 24.789084 8.324431 3472 7.891421
671131 24.839506 8.314724 3477 7.890776
671133 24.869759 8.305223 3480 7.890142
671135 24.910097 8.295927 3484 7.889522
671137 24.950434 8.286835 3488 7.888916
671139 24.990772 8.277946 3492 7.888323
671141 25.021025 8.269261 3495 7.887742
671143 25.061362 8.260777 3499 7.887175
671145 25.091616 8.252494 3502 7.886620
671147 25.131953 8.244413 3506 7.886080
671149 25.162206 8.236531 3509 7.885551
671151 25.192460 8.228850 3512 7.885036
671153 25.222713 8.221367 3515 7.884534
671155 25.252966 8.214084 3518 7.884045
671157 25.283219 8.206999 3521 7.883570
671159 25.313472 8.200112 3524 7.883107
671161 25.343725 8.193422 3527 7.882657
671163 25.373978 8.186930 3530 7.882221
671165 25.394147 8.180635 3532 7.881797
671167 25.424400 8.174538 3535 7.881387
671169 25.444569 8.168637 3537 7.880989
671171 25.464738 8.162932 3539 7.880604
671173 25.494991 8.157423 3542 7.880233
671175 25.515160 8.152110 3544 7.879875
671177 25.535328 8.146993 3546 7.879529
671179 25.555497 8.142071 3548 7.879197
671181 25.575666 8.137344 3550 7.878878
671183 25.595834 8.132812 3552 7.878572
671185 25.616003 8.128475 3554 7.878280
671187 25.626088 8.124333 3555 7.877999
671189 25.646256 8.120385 3557 7.877733
671191 25.656341 8.116631 3558 7.877478
671193 25.676509 8.113071 3560 7.877238
671195 25.686594 8.109705 3561 7.877010
671197 25.696678 8.106533 3562 7.876796
671199 25.716847 8.103554 3564 7.876595
671201 25.726931 8.100768 3565 7.876407
671203 25.737016 8.098175 3566 7.876231
671205 25.747100 8.095776 3567 7.876069
671207 25.757184 8.093569 3568 7.875921
671209 25.757184 8.091555 3568 7.875784
671211 25.777353 8.089734 3570 7.875662

One diving log of mapping water density from dynamics, STD and STD with 15% salinity error

time 1695747 depth 1.341396 density 1005.411579 desity_std 1003.738162 s_error 1004.510959
time 1695751 depth 1.835592 density 1004.363920 desity_std 1003.741102 s_error 1004.513900
time 1695755 depth 2.390300 density 1004.025284 desity_std 1003.749201 s_error 1004.522033
time 1695759 depth 2.965178 density 1003.343738 desity_std 1003.757641 s_error 1004.530509
time 1695763 depth 3.529969 density 1003.685242 desity_std 1003.766083 s_error 1004.538988
time 1695767 depth 4.094758 density 1003.859838 desity_std 1003.774520 s_error 1004.547460
time 1695771 depth 4.669630 density 1003.685269 desity_std 1003.784827 s_error 1004.557804
time 1695775 depth 5.234416 density 1003.685283 desity_std 1003.796420 s_error 1004.569434
time 1695779 depth 5.799200 density 1003.685297 desity_std 1003.807962 s_error 1004.581014
time 1695783 depth 6.363982 density 1003.685310 desity_std 1003.819454 s_error 1004.592542
time 1695787 depth 6.928763 density 1003.685324 desity_std 1003.827584 s_error 1004.600735
time 1695791 depth 7.493541 density 1003.509251 desity_std 1003.835625 s_error 1004.608838
time 1695795 depth 8.048233 density 1003.685351 desity_std 1003.843600 s_error 1004.616876
time 1695799 depth 8.613009 density 1003.509279 desity_std 1003.853251 s_error 1004.626649
time 1695803 depth 9.167698 density 1003.685379 desity_std 1003.865793 s_error 1004.639419
time 1695807 depth 9.722385 density 1004.025446 desity_std 1003.878277 s_error 1004.652132
time 1695811 depth 10.277070 density 1004.025458 desity_std 1003.890704 s_error 1004.664786
time 1695815 depth 10.831754 density 1003.847921 desity_std 1003.909640 s_error 1004.684158
time 1695819 depth 11.386436 density 1003.509347 desity_std 1003.931169 s_error 1004.706209
time 1695823 depth 11.931031 density 1003.847946 desity_std 1003.952451 s_error 1004.728013
time 1695827 depth 12.475625 density 1003.847958 desity_std 1003.973535 s_error 1004.749616
time 1695831 depth 13.020218 density 1003.847970 desity_std 1003.994420 s_error 1004.771018
time 1695835 depth 13.554724 density 1004.185140 desity_std 1004.023555 s_error 1004.801030
time 1695839 depth 14.089228 density 1004.185151 desity_std 1004.054615 s_error 1004.833063
time 1695843 depth 14.623731 density 1004.004643 desity_std 1004.085220 s_error 1004.864635
time 1695847 depth 15.148147 density 1004.004654 desity_std 1004.115326 s_error 1004.895702
time 1695851 depth 15.672562 density 1004.004665 desity_std 1004.144983 s_error 1004.926317
time 1695855 depth 16.196976 density 1003.822652 desity_std 1004.174199 s_error 1004.956483
time 1695859 depth 16.711303 density 1003.822663 desity_std 1004.206634 s_error 1004.990132
time 1695863 depth 17.215544 density 1004.155465 desity_std 1004.243195 s_error 1005.028241
time 1695867 depth 17.719783 density 1004.155474 desity_std 1004.279086 s_error 1005.065669
time 1695871 depth 18.224022 density 1003.970489 desity_std 1004.314314 s_error 1005.102422
time 1695875 depth 18.718174 density 1003.970499 desity_std 1004.348841 s_error 1005.138462
time 1695879 depth 19.202240 density 1004.486860 desity_std 1004.382678 s_error 1005.173801
time 1695883 depth 19.696390 density 1004.112440 desity_std 1004.415930 s_error 1005.208542
time 1695887 depth 20.170369 density 1004.300415 desity_std 1004.450319 s_error 1005.244683
time 1695891 depth 20.644347 density 1004.628877 desity_std 1004.484768 s_error 1005.280974
time 1695895 depth 21.118324 density 1004.439449 desity_std 1004.518566 s_error 1005.316597
time 1695899 depth 21.582215 density 1004.439457 desity_std 1004.551675 s_error 1005.351515
time 1695903 depth 22.046105 density 1004.248525 desity_std 1004.584155 s_error 1005.385787
time 1695907 depth 22.499910 density 1004.248532 desity_std 1004.615968 s_error 1005.419377
time 1695911 depth 22.943628 density 1004.765024 desity_std 1004.647127 s_error 1005.452296
time 1695915 depth 23.397430 density 1004.378707 desity_std 1004.677735 s_error 1005.484649
time 1695919 depth 23.831063 density 1004.572628 desity_std 1004.707664 s_error 1005.516307

time 1695923 depth 24.264694 density 1004.895268 desity_std 1004.737019 s_error 1005.547374
time 1695927 depth 24.698324 density 1004.699889 desity_std 1004.764897 s_error 1005.576975
time 1695931 depth 25.121869 density 1004.699895 desity_std 1004.790803 s_error 1005.604628
time 1695935 depth 25.545413 density 1004.699901 desity_std 1004.816290 s_error 1005.631847
time 1695939 depth 25.968956 density 1004.503019 desity_std 1004.841367 s_error 1005.658640
time 1695943 depth 26.382413 density 1004.503025 desity_std 1004.865994 s_error 1005.684967
time 1695947 depth 26.785786 density 1005.019635 desity_std 1004.890176 s_error 1005.710836
time 1695951 depth 27.199242 density 1004.821280 desity_std 1004.914016 s_error 1005.736346
time 1695955 depth 27.592528 density 1005.138086 desity_std 1004.937377 s_error 1005.761363
time 1695959 depth 27.995898 density 1004.621428 desity_std 1004.960408 s_error 1005.786034
time 1695963 depth 28.389183 density 1004.621433 desity_std 1004.983019 s_error 1005.810271
time 1695967 depth 28.772383 density 1004.936762 desity_std 1005.005217 s_error 1005.834079
time 1695971 depth 29.155582 density 1004.936767 desity_std 1005.027054 s_error 1005.857512
time 1695975 depth 29.538780 density 1004.936771 desity_std 1005.048536 s_error 1005.880576
time 1695979 depth 29.921978 density 1004.733937 desity_std 1005.069668 s_error 1005.903275
time 1695983 depth 30.295090 density 1004.733942 desity_std 1005.090410 s_error 1005.925569
time 1695987 depth 30.658118 density 1005.046347 desity_std 1005.110786 s_error 1005.947505
time 1695991 depth 31.021145 density 1005.046351 desity_std 1005.130983 s_error 1005.969399
time 1695995 depth 31.384172 density 1005.046354 desity_std 1005.150876 s_error 1005.990973
time 1695999 depth 31.747197 density 1004.840544 desity_std 1005.170469 s_error 1006.012230
time 1696003 depth 32.100138 density 1004.840548 desity_std 1005.189718 s_error 1006.033127
time 1696007 depth 32.442995 density 1005.150026 desity_std 1005.208627 s_error 1006.053668
time 1696011 depth 32.795934 density 1004.840556 desity_std 1005.227297 s_error 1006.073952
time 1696015 depth 33.128706 density 1005.458055 desity_std 1005.245588 s_error 1006.093842
time 1696019 depth 33.471560 density 1005.150035 desity_std 1005.263646 s_error 1006.113483
time 1696023 depth 33.804330 density 1005.458060 desity_std 1005.281381 s_error 1006.132784
time 1696027 depth 34.137099 density 1005.247797 desity_std 1005.298843 s_error 1006.151797
time 1696031 depth 34.459784 density 1005.247800 desity_std 1005.315990 s_error 1006.170478
time 1696035 depth 34.782469 density 1005.247802 desity_std 1005.332872 s_error 1006.188878
time 1696039 depth 35.105153 density 1005.247805 desity_std 1005.349494 s_error 1006.207002
time 1696043 depth 35.417752 density 1005.552897 desity_std 1005.365811 s_error 1006.224805
time 1696047 depth 35.730351 density 1005.552899 desity_std 1005.381874 s_error 1006.242338
time 1696051 depth 36.042949 density 1005.339659 desity_std 1005.397687 s_error 1006.259606
time 1696055 depth 36.345463 density 1005.339661 desity_std 1005.413206 s_error 1006.276564
time 1696059 depth 36.647977 density 1005.339663 desity_std 1005.428483 s_error 1006.293263
time 1696063 depth 36.950490 density 1005.124923 desity_std 1005.443639 s_error 1006.309973
time 1696067 depth 37.242919 density 1005.124926 desity_std 1005.458550 s_error 1006.326441
time 1696071 depth 37.535348 density 1005.124928 desity_std 1005.473251 s_error 1006.342681
time 1696075 depth 37.817692 density 1005.425606 desity_std 1005.487694 s_error 1006.358645
time 1696079 depth 38.100036 density 1005.425607 desity_std 1005.501930 s_error 1006.374385
time 1696083 depth 38.382379 density 1005.425609 desity_std 1005.515961 s_error 1006.389902
time 1696087 depth 38.654639 density 1005.724828 desity_std 1005.529742 s_error 1006.405150
time 1696091 depth 38.926898 density 1005.724829 desity_std 1005.543321 s_error 1006.420179
time 1696095 depth 39.199156 density 1005.505634 desity_std 1005.556703 s_error 1006.434992
time 1696099 depth 39.461331 density 1005.505636 desity_std 1005.569839 s_error 1006.449543
time 1696103 depth 39.723505 density 1005.505637 desity_std 1005.582781 s_error 1006.463882
time 1696107 depth 39.985679 density 1005.505638 desity_std 1005.595530 s_error 1006.478010
time 1696111 depth 40.237769 density 1005.801916 desity_std 1005.608041 s_error 1006.491882
time 1696115 depth 40.489858 density 1005.801917 desity_std 1005.620362 s_error 1006.505547
time 1696119 depth 40.741948 density 1005.579745 desity_std 1005.632497 s_error 1006.519007
time 1696123 depth 40.983953 density 1005.579746 desity_std 1005.644398 s_error 1006.532217
time 1696127 depth 41.225958 density 1005.579747 desity_std 1005.656117 s_error 1006.545227
time 1696131 depth 41.467963 density 1005.579748 desity_std 1005.667654 s_error 1006.558037
time 1696135 depth 41.699884 density 1005.873082 desity_std 1005.678963 s_error 1006.570602
time 1696139 depth 41.931804 density 1005.873083 desity_std 1005.690094 s_error 1006.582971
time 1696143 depth 42.163725 density 1005.647935 desity_std 1005.701049 s_error 1006.595147
time 1696147 depth 42.385561 density 1005.647936 desity_std 1005.711781 s_error 1006.607083
time 1696151 depth 42.607398 density 1005.647936 desity_std 1005.722341 s_error 1006.618828
time 1696155 depth 42.829234 density 1005.421292 desity_std 1005.732635 s_error 1006.630309
time 1696159 depth 43.040986 density 1005.421293 desity_std 1005.742532 s_error 1006.641409
time 1696163 depth 43.252738 density 1005.421294 desity_std 1005.752280 s_error 1006.652343
time 1696167 depth 43.464490 density 1005.421295 desity_std 1005.761881 s_error 1006.663111
time 1696171 depth 43.666158 density 1005.710202 desity_std 1005.771288 s_error 1006.673669
time 1696175 depth 43.867826 density 1005.710202 desity_std 1005.780550 s_error 1006.684063
time 1696179 depth 44.069494 density 1005.480582 desity_std 1005.789669 s_error 1006.694296
time 1696183 depth 44.261079 density 1005.480583 desity_std 1005.798596 s_error 1006.704320
time 1696187 depth 44.452663 density 1005.480584 desity_std 1005.807381 s_error 1006.714184
time 1696191 depth 44.644246 density 1005.480585 desity_std 1005.816025 s_error 1006.723889
time 1696195 depth 44.825747 density 1005.766543 desity_std 1005.824480 s_error 1006.733387
time 1696199 depth 45.007247 density 1005.766543 desity_std 1005.832795 s_error 1006.742727
time 1696203 depth 45.188747 density 1005.766544 desity_std 1005.840970 s_error 1006.751911
time 1696207 depth 45.360163 density 1006.051033 desity_std 1005.848958 s_error 1006.760889